uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Telemática
2016/2017

Trabajo Fin de Grado

Service Function Chaining en NFV: Evaluación práctica con OpenStack

---

Javier Bautista Ramos

Tutor:
Carlos Jesus Bernardos Cano
Director:
Pablo Serrano Yáñez-Mingot

Resumen

Actualmente el crecimiento del sector de las telecomunicaciones en internet lleva asociado el despliegue de una gran infraestructura. Esta infraestructura se compone de dispositivos de red que permiten funciones como pueden ser routers, firewalls para el bloqueo de tráfico, controles parentales, sistemas de inspección de paquetes como IDS o IPS o incluso servidores físicos.

Con el avance de las tecnologías, las empresas distribuidoras de dispositivos, han avanzado en el mercado ofreciendo nuevas soluciones, un ejemplo claro son los firewalls, que desde su aparición en 1988, han pasado por tres generaciones, estando presentes no solo en dispositivos físicos, sino también incluidos en sistemas finales. En el mismo caso se sitúan los routers, que han ido evolucionando desde su aparición en ARPANET, por entonces conocidos como IMP (Interface Message Processor), hasta hoy en día, donde ya funcionan con un hardware específico para poder acelerar las funciones de encaminamiento, o la encriptación de paquetes con IPSec. En el caso de los routers, el avance era necesario en muchos aspectos, ya que los protocolos de comunicaciones avanzaban a pasos agigantados, introduciendo tecnologías como RIP u OSPF, las cuales necesitan de dispositivos preparados para su funcionamiento, a su vez también, un caso más reciente es la transición a IPV6, donde los dispositivos actuales en gran parte no la soportan, por lo tanto, en un caso tan extremo como el funcionamiento del protocolo IP, es necesario el cambio del hardware en todos los usuarios conectados.

Cada cierto tiempo, las empresas se ven obligadas a la actualización de sus dispositivos de red físicos, para así poder adecuarse a los nuevos estándares, la demanda de los usuarios en calidad de servicio o incluso el crecimiento interno de la empresa y sus empleados. El tiempo de vida de los dispositivos fijos es por lo general largo, debido al alto coste del despliegue de estos, por lo que su actualización requiere de una nueva inversión ya que no es posible hacer actualizaciones sobre el hardware ya existente.

Es por esto que las tecnologías de virtualización emergentes, solucionan los problemas descritos anteriormente, de manera que es posible la simulación de las capacidades de muchos dispositivos de red tradicionales, en un entorno virtualizado, donde no sea necesario el despliegue de nuevos elementos y se puedan implementar varias soluciones de red dentro de una misma plataforma de virtualización.

Así, la inversión que necesita una empresa para desplegar todo su entorno de red es única, es decir, basta con adquirir varios servidores con las características adecuadas, para así poder desplegar sobre estos una plataforma Cloud donde poder crear todas las funciones de red virtualizadas. Si en un futuro se necesita actualizar o cambiar una función de red por otra que ofrezca un mejor servicio, será tan simple como desarrollar sus capacidades a través de ingeniería del software, o adquirirla a empresas que comercialicen este software, siempre a un menor precio que el dispositivo de red en su versión física.

En la actualidad ya existen empresas que comercializan soluciones Cloud, tanto el modelo IaaS (Infrastructure as a Service), PaaS (Platform as a Service) y SaaS (Software as a Service), como pueden ser Amazon con su plataforma AWS o Google con Google Cloud. Muchas empresas deciden alojar sus servicios tales como bases de datos o servicios web dentro de estos entornos Cloud, pero estas soluciones no se ajustan a la virtualización de dispositivos de red dentro del propio entorno empresarial, a no ser que se disponga de una nube privada, como es el caso de Amazon o Google con sus respectivas nubes, las cuales les pueden servir para virtualizar todos sus dispositivos. Por lo tanto la solución idónea para una empresa que quiera pasarse al nuevo paradigma Cloud, con todas las ventajas que conlleva descritas anteriormente, es la de desplegar su propia nube privada. Para esto existen varias plataformas de despliegue, pero la más utilizada y apoyada a día de hoy es la de OpenStack, un entorno Cloud Open Source en el que trabajan empresas como IBM o AT&T.

A todas las ventajas anteriormente comentadas, hay que unirle el uso de nueva arquitectura de encaminamiento de paquetes a través de las funciones de red, Service Function Chaining (SFC), esta nueva arquitectura fue planteada hace cuatro años, y consiste en el encaminamiento de los paquetes a través de las distintas funciones de red virtualizadas según se requiera, esto significa que no es necesario encaminar páquetes que pertenezcan a una llamada de voz IP a través de un sistema IDS (Intrusion Detection System) o paquetes de tráfico HTTP a través de un acelerador de vídeo. que en un entorno de red tradicional, pueden estar desde el punto de vista de red, topologicamente seguidos. Así en un entorno virtualizado, se puede elegir por que funciones de red tiene que pasar cierto tipo de tráfico, para así optimizar la entrega de paquetes.

Gracias a las tecnologías de virtualización y en concreto a las recientes arquitecturas de Network Function Virtualization (NFV) y Software Defined Networks (SDN), es posible la virtualización de funciones de red (e.g. Firewall, Control parental, IDS) en servidores, a la vez que es posible la virtualización de redes sobre las que van conectadas dichas funciones.

SDN es un conjunto de técnicas directamente relacionadas con las redes de comunicaciones en Internet, que tiene como objetivo la implantación de servicios de red tradicionales mediante la separación del plano de control y de datos. Dicho de otra manera, SDN permite la creación de entornos de red dinámicos, escalables a través de interfaces abiertas que permiten simular completamente el comportamiento de los dispositivos tradicionales estáticos en cuanto a interconectividad se refiere.

La separación del plano de control del plano de datos, permite que las decisiones sobre el tráfico de red, el envío los paquetes según los criterios correspondientes, se lleven a cabo a través del plano de control (controlador SDN), así es posible decidir el camino del tráfico de red de una manera u otra sin tener que tocar ningún dispositivo de red físico. Por otra parte, las capas inferiores que se encargan del envío entre dispositivos virtuales de los paquetes, es el plano de datos.

Para poder interconectar estos dos planos, aparece el protocolo OpenFlow, que se encarga de comunicar las decisiones del plano de control al plano de datos, para virtualizar el switch que se encarga de conectar los dispositivos virtuales, se introduce Open vSwitch en el plano de datos, de esta manera se consigue una conectividad total entre las funciones virtualizadas como si fuera un entorno de red tradicional.

Network Function Virtualization (NFV), es el marco de referencia que se utiliza para la virtualización de las funciones de red, y que permite que el software desarrollado se despliegue en diferentes plataformas. Este software puede ser desarrollado por diferentes proveedores, pero no es incompatible, por lo tanto la plataforma sobre la que se despliegue, ofrece la ventaja de soluciones multivendor. Para la creación de máquinas virtualizadas, se usa OpenStack, una nube privada similar a AWS o Google Cloud. Gracias a OpenStack, es posible utilizar las tecnologías SDN y NFV conjuntamente, para así poder desplegar funciones de red interconectadas dentro de redes virtualizadas, y con conexión a redes externas (internet).

OpenStack es una solución que se puede implementar de varias maneras, así, puede ser desplegado sobre varios nodos de red según las necesidades, o sobre un único nodo. Se compone de varios módulos, los cuales se encargan cada uno de una función específica (networking, almacenamiento de imágenes, orquestación, etc.). En muchos despliegues no es necesario la instalación de todos los módulos, ya que algunos no son indispensables. En este proyecto se opta por la instalación en un único nodo mediante DevStack, una serie de scripts para la instalación de OpenStack en entornos de prueba.

Para la instalación de OpenStack también se propondrán las diferentes soluciones posibles, y una explicación sobre la decisión final y sus ventajas frente a la otra. Esto quiere decir que la solución que ofrece este proyecto no es ni mucho menos única, demostrando así la versatilidad que ofrece y las capacidades según las necesidades y material disponible.

Para la creación de cadenas de servicio, o dicho de otra manera, cadenas donde los paquetes recorren diferentes funciones de red en función del tipo de tráfico, se utiliza SFC. Esta arquitectura está siendo actualmente desarrollada por varias empresas del sector, por lo que no es una versión final y contiene algunos fallos, pero la funcionalidad principal es estable y se puede implementar. SFC utiliza una encapsulación en los paquetes que contiene la información sobre la cadena que debe seguir, ya que pueden existir varias, también introduce nuevos dispositivos virtualizados que se encargan de dirigir los paquetes entre las diferentes funciones de red.

En la implementación de SFC existen varias posibilidades, en este proyecto se ha optado por el uso del plugin de OpenStack, que usa como controlador SDN el propio que ofrece el módulo de networking de OpenStack, ya que es suficiente para mostrar todas las capacidades de las que dispone SFC, también se explican los motivos de la decisión y otras alternativas que existen.

En el proyecto se evaluarán diferentes escenarios de red, cada uno de ellos muestra las funcionalidades de SFC y sus ventajas respecto a los entornos de red tradicionales. Se explicará paso por paso con scripts, como llegar a los diferentes escenarios y se mostrarán las rutas que siguen los paquetes del tráfico generado.

Para la instalación de OpenStack, en este proyecto se ha decidido el uso del Cloud que ofrece Google, para así poder apreciar las ventajas que ofrece la virtualización frente a la adquisición de un nodo físico. También se mostrará como instalar OpenStack a través de DevStack y la preparación del entorno de Google Cloud. Se mostrará un pequeño estudio con los costes asociados a alquilar un servidor en la nube frente a adquirir un pequeño servidor físico, la planificación seguida durante todo el proyecto y los costes totales del mismo. Por último se indicarán posibles desarrollos y líneas de investigación futuras relacionadas con la tecnología de SFC.

Abstract

Currently, the growth of the telecommunications sector on the internet is associated with the deployment of a large infrastructure. This infrastructure consists of network devices that allow functions such as routers, firewalls for traffic blocking, parental controls, packet inspection systems such as IDS or IPS or even physical servers.

With the advancement of technologies, device distributors have advanced in the market offering new solutions, a clear example are the firewalls, which since its appearance in 1988 have gone through three generations, being present not only in physical devices, but also included in end systems. In the same case the routers are located, which have evolved since its appearance in ARPANET, then known as IMP (Interface Message Processor), until today, where they already work with specific hardware to be able to accelerate the functions of routing, or packet encryption with IPSec. In the case of routers, the advancement was necessary in many aspects, since the communication protocols were advancing by leaps and bounds, introducing routing technologies like RIP or OSPF, which need devices ready to operate, as well as the recent case of transition to IPV6, where the current devices largely do not support it, therefore, in a case as extreme as the operation of the IP protocol, it is necessary to change the hardware in all connected users.

From time to time, companies are forced to update their physical network devices, in order to be able to adapt to the new standards, the demand of the users in quality of service or even the internal growth of the company and its employees. The lifespan of the fixed devices is usually long, due to the high cost of the deployment of these, so that their update requires a new investment as it is not possible to make updates on the existing hardware.

This is why emerging virtualization technologies solve the problems described above, so that it is possible to simulate the capabilities of many traditional network devices in a virtualized environment, where the deployment of new physical elements is not necessary and can be deployed multiple network solutions within the same virtualization platform.

Thus, the investment that a company needs to deploy its entire network environment is unique, ie, it is enough to acquire several servers with the appropriate characteristics, in order to be able to deploy on these a Cloud platform, where to be able to create all the virtualized network functions. If a network function needs to be updated or changed in the future for a better service, it will be as simple as developing its capabilities through software engineering, or acquiring it to companies that market this software, at a lower price than the network device in its physical version.

There are already companies that market Cloud solutions, the IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service) models, such as Amazon with its AWS platform or Google with Google Cloud. Many companies choose to host their services such as databases or web services within these cloud environments, but these solutions do not fit the virtualization of network devices within the business environment, unless a private cloud is available, as is the case of Amazon or Google with their respective clouds, which can serve to virtualize all their devices. Therefore the ideal solution for a company that wants to move to the new Cloud paradigm, with all the advantages that it entails described above, is to deploy its own private cloud. For this there are several deployment platforms, but the most used and supported today is OpenStack, a Cloud Open Source environment supported by companies such as IBM or AT&T.

To all the advantages mentioned above, the use of a new packet routing architecture through the network functions, Service Function Chaining (SFC), has been added. This new architecture was proposed four years ago, and consists of the routing of the packets across the various virtualized network functions as required, this means that it is not necessary to route packets belonging to an IP voice call through an IDS (Intrusion Detection System) system or HTTP traffic packets through a video accelerator. From the point of view of a tradicional network environment, those functions are topologically followed. So in a virtualized environment, you can choose by which network functions you have to pass certain type of traffic, in order to optimize the delivery of packages and improve quality of service.

Thanks to virtualization technologies and in particular to the recent Network Function Virtualization (NFV) and Software Defined Networks (SDN) architectures, it is possible to virtualize network functions (eg Firewall, Parental Control, IDS) in servers, and at the same time, it is possible to interconnect those functions.

SDN is a set of techniques directly related to Internet communications networks, which aims to implement traditional network services by separating the control plane and data plane. Put another way, SDN allows the creation of dynamic, scalable network environments through open interfaces that allow to fully simulate the behavior of traditional static devices in terms of interconnectivity.

The separation of the control plane from the data plane, allows the decisions on the network traffic and sending the packets according to the corresponding criteria, carried out through the control plane (controller SDN), so it is possible to decide the path of the network traffic in one way or another without having to touch any physical network device. On the other hand, the lower layers that are responsible for sending the packets between virtual devices of the packages, is the data plane.

In order to interconnect these two planes, the OpenFlow protocol appears, which is responsible for communicating the decisions of the control plane to the data plane. In order to virtualize the switch that connects virtual devices, Open vSwitch is introduced in the data plane, thus achieving full connectivity between virtualized functions as if it were a traditional network environment.

Network Function Virtualization (NFV) is the framework that is used for virtualization of network functions, and allows software developed representing these functions, to be deployed on different platforms. This software can be developed by different vendors, but it is not incompatible, therefore the platform on which it is deployed, offers the advantage of multivendor solutions.

For the creation of virtualized machines, OpenStack, a private cloud similar to AWS or Google Cloud, is used. Thanks to OpenStack, it is possible to use SDN and NFV technologies together, so that you can deploy network functions interconnected within virtualized networks, and with connection to external networks (Internet).

OpenStack is a solution that can be deployed in several ways, so it can be deployed over multiple network nodes as needed, or over a single node. It consists of several modules, each of which is responsible for a specific function (networking, image storage, orchestration, etc.). In many deployments it is not necessary to install all the modules, since some are not indispensable. In this project we opt for the installation in a single node through DevStack, a series of scripts for the installation of OpenStack in test environments.

For the installation of OpenStack will also be proposed different possible solutions, and an explanation about the final decision and its advantages over the other. This means that the solution offered by this project is by no means unique, thus demonstrating the versatility it offers and the capabilities according to the needs and material available.

For the creation of service chains, or in other words, chains where the packets are routed across different network functions depending on the type of traffic, SFC is used. This architecture is currently being developed by several companies in the sector, so it is not a final version and contains some bugs, but the main functionality is stable and can be implemented. SFC uses an encapsulation in the packets containing the information about the chain to follow, since there may be several, also introduces new virtualized devices that are responsible for directing the packets between the different network functions.

In the implementation of SFC there are several possibilities, in this project the OpenStack plugin has been chosen, which uses the SDN driver that OpenStack networking module offers, as it is sufficient to show all the capabilities that SFC has, the reasons for the decision and other alternatives are explained as well.

The project will evaluate different network scenarios, each of which shows the functionalities of SFC and its advantages over traditional network environments. It will be explained step by step with scripts, how to get to the different scenarios and will show the routes that follow the packets of the generated traffic.

For the installation of OpenStack, in this project has been decided to use the Cloud offered by Google, in order to appreciate the advantages of virtualization versus the acquisition of a physical node. It will also show how to install OpenStack through DevStack and the preparation of the Google Cloud environment. Will be shown a small study with the costs associated with renting a server in the cloud versus acquiring a small physical server, the planning followed throughout the project and the total costs of it. Finally, possible future developments and lines of research related to SFC technology will be indicated.

Índice general

Índice de figuras

Índice de tablas

Glosario

La lista de los acrónimos utilizados en este proyecto es la siguiente:

NFV Network Function Virtualization

SDN System Defined Network

SFC Service Function Chaining

FW Firewall

IDS Intrusion Detection System

IPS Intrusion Prevention System

VM Virtual Machine

IaaS Infrastructure as a Service

SaaS Software as a Service

PaaS Platform as a Service

CPE Customer Premise Equipment

VNF Virtualized Network Function

VoIP Voice over Internet Protocol

IoT Internet of Things

NIST National Institute of Standards and Technology

DPI Deep Packet Inspector

CLI Command Line Interface

SF Service Function

PP Port Pair

PPG Port Pair Group

SFF Service Function Forwarder

SFP Service Function Path

ODL OpenDayLight

Capítulo 1

Introducción

1.1. Motivación del proyecto

Cada año que pasa son más las empresas que deciden disponer de elementos de red tales como firewalls, sistemas IDS (Intrusion Detection System) para la seguridad, dispositivos de función de red como switches, routers o CPEs (Customer Premises Equipment). La arquitectura relacionada con las redes tradicionales es una arquitectura estática, donde tanto Hardware como Software suelen estar limitados y se integran de una forma predeterminada en un punto de la red.

NFV es una arquitectura de virtualización que comenzó a desarrollarse en el año 2012 por miembros del ETSI en Alemania, su principal objetivo es el de virtualizar los dispositivos de red tradicionales, tales como switches, routers o firewalls. Esta virtualización es posible realizarla en diferentes plataformas, permitiendo multiples puntos de gestión de la red, además de que estas plataformas soportan soluciones multivendor (multiples fabricantes).

Para poder llegar a esta solución, se deben crear los elementos de función de red virtualizados (VNF sus siglas en inglés), estos elementos realizaran una función de red (firewall, switch, router,...), y mediante la combinación de todos ellos se conseguirá implementar conjunto de dispositivos de red virtualizados. Así, de esta manera, se consigue sustituir elementos de la red complejos que trabajan sobre un hardware único y genérico, por elementos virtualizados sobre plataformas compartidas capaces de compartir recursos de hardware.

Por otro lado, la arquitectura SFC, nos permite el encaminamiento de paquetes a través diferentes elementos de la red, de forma ordenada y selectiva, esto nos permite la creación de rutas a través de la red diferentes según el tipo de tráfico que se esté cursando. SFC usa las capacidades de SDN para su correcto funcionamiento y junto con las capacidades que ofrece NFV, es posible un uso más eficiente de la red para cada tipo de sesión que esté activa. Por ejemplo, una sesión de video o VoIP tiene menos demanda que un simple acceso a tráfico web, el uso de SFC permite habilitar para estas sesiones los recursos adecuados para asegurar su correcta transmisión (ancho de banda, encriptación, QoS).

Estos hechos suponen una gran motivación en la realización de este proyecto, dado que una correcta arquitectura de red usando estas técnicas le virtualización supone un ahorro considerable en el entorno empresarial y puede suponer la creación de nuevas empresas pequeñas o start-ups que no disponen de esta inversión inicial.

En este proyecto se pretende desplegar OpenStack sobre un único servidor dedicado alojado en el cloud que ofrece Google, de esta manera será posible la virtualización de nuevas instancias y la creación de nuevos proyectos y usuarios que puedan acceder a estos. Seguidamente se probarán distintos escenarios de red usando SFC, para probar su correcto funcionamiento.

En este proyecto se pretende evaluar las funcionalidades que aporta SFC a, mediante diferentes escenarios de pruebas. En estos escenarios se podrán apreciar las principales características que tiene SFC, entre las que destacan, encaminamiento de paquetes a través de funciones o balanceo de carga entre funciones de red del mismo o diferente tipo, esto aporta una solución diferente y más efectiva que los dispositivos de red tradicionales, donde los paquetes no pueden ser separados en función del tráfico y no se asegura la calidad de servicio.

Para el despliegue de la solución se propone la instalación de OpenStack, ya que es un entorno Cloud OpenSource, el cúal permite la creación de máquinas virtuales en línea y la conexión entre éstas gracias a las tecnologías SDN y NFV. OpenStack será instalado sobre el Google Cloud, para así comprobar de primera mano las ventajas y el ahorro en costes que produce una solución Cloud ya implantada en el mercado.

1.2. Objetivos del proyecto

Con la realización de este proyecto, se pretende conseguir la creación de entornos de red con diferentes funciones de red, en los cuales el enrutado de paquetes sea correcto acorde a las reglas de SFC. De esta manera se habrá conseguido hacer una evaluación práctica exitosa de Service Function Chaining y sus aplicaciones.

El primer objetivo consiste en la instalación de OpenStack en el servidor Google Cloud, una vez instalado, se procederá a describir las principales características de éste, describiendo el uso del cliente web, instanciación de máquinas virtuales y reglas del firewall. Tiene los siguientes hitos:

- Creación de una instancia con Ubuntu 16.04 en Google Cloud.
- Instalación de OpenStack usando Devstack.
- Como usar OpenStack.
    - Uso del Dashboard y la línea de comandos.
    - Manejo de proyectos y usuarios.

  - Operaciones incluyendo manejo del local.conf, imágenes, IPs flotantes, conexiones a las instancias.

- Manejo de grupos de seguridad.

Una vez se hayan cumplido los objetivos del uso básico de OpenStack, para poder evaluar el funcionamiento del SFC, será necesario crear varias instancias, asignar un rol a cada una, crear los puertos asociados a éstas por los cuales irán los paquetes de red que deban seguir las cadenas de funciones, configurar las instancias para habilitar el forwarding de paquetes y activar las interfaces de red :

- Línea de comandos de *Neutron*[1] para el uso de SFC.

- Escenarios de red usando SFC para encadenar las funciones y verificar la ruta de los paquetes.

1.3. Entorno socio-económico

La idea de este proyecto es dar una solución para la posible migración de dispositivos de red en entornos empresariales que hagan uso de ellos, a servicios de virtualización los cuales son notablemente más reutilizables, flexibles, de fácil manejo y baratos [4] que las soluciones estándar que proponen los fabricantes de equipamiento de telecomunicaciones. Si combinamos lo anterior con la idea que proporciona SFC, podemos formar un entorno en el cual la optimización de los recursos sea la adecuada, dando a cada tipo de sesión de red los recursos y los filtros que necesita, pudiendo así optimizar el ancho de banda, la encriptación o calidad de servicio, ofreciendo así una mejor experiencia al cliente final y a su vez, pudiendo aumentar el número de estos. En la figura 1.1[2] se muestra un ejemplo de rendimiento de SFC.

Por ejemplo, suponiendo que cierta empresa distribuidora de vídeo bajo demanda, la cual, por el constante crecimiento del sector audiovisual y por consiguiente de sus clientes, se ve obligada a el despliegue de una nueva infraestructura de seguridad y la ampliación de recursos en sus servidores dedicados para así poder ofrecer a sus usuarios la experiencia de video que ellos reclaman. Si esta empresa decide renovar por completo sus servidores y comprar soluciones firewall o controles parentales, se encontrará ante una fuerte inversión sin posiblemente haber recuperado la que hizo al principio para desplegar su infraestructura inicial, y posiblemente teniendo que renovar de nuevo todo el despliegue dentro de unos años.

[1] Neutron es el módulo de networking en OpenStack
[2] http://openvswitch.org/support/ovscon2015/17/1310-nakagawa.pdf

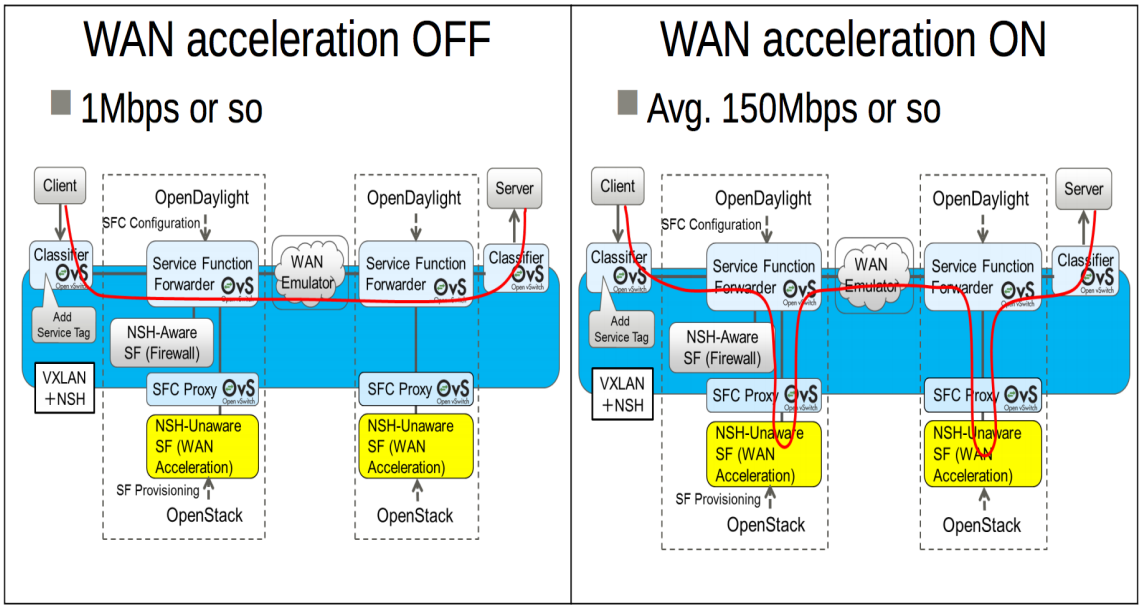


Figura 1.1: Marco de pruebas de SFC en Fujitsu [30]

Las soluciones SDN y NFV permitirían que esta empresa pudiera montar toda su infraestructura únicamente en servidores de gran potencia los cuales soportan soluciones de múltiples proveedores, tendrían la posibilidad de usar la arquitectura SFC para tratar cada tipo de tráfico de una manera selectiva y optimizar así todos sus recursos. Este mismo ejemplo se podría aplicar a cualquier tipo de empresa que ofrezca servicios a través de Internet, véase servicios de VoIP, servicios cloud, o ISPs.

En definitiva, el uso combinado de estas tres arquitecturas puede llegar a tener un gran valor en el entorno empresarial ya que todo indica a que en un futuro próximo la demanda de servicios a traves de Internet crezca notoriamente (IoT, Big Data, Ciberseguridad...) y se necesita de soluciones diferentes a las tradicionales que proporcionan hoy en día los proveedores. Como se muestra en la tabla 1.1[3], los ahorros del uso de SDN son considerables.

[3]http://searchtelecom.techtarget.com/photostory/2240177303/SDN-survey-offers-snapshot-of-provider-plans-and-vendor-issues/5/SDN-architecture-expected-to-bring-simplicity-cost-savings

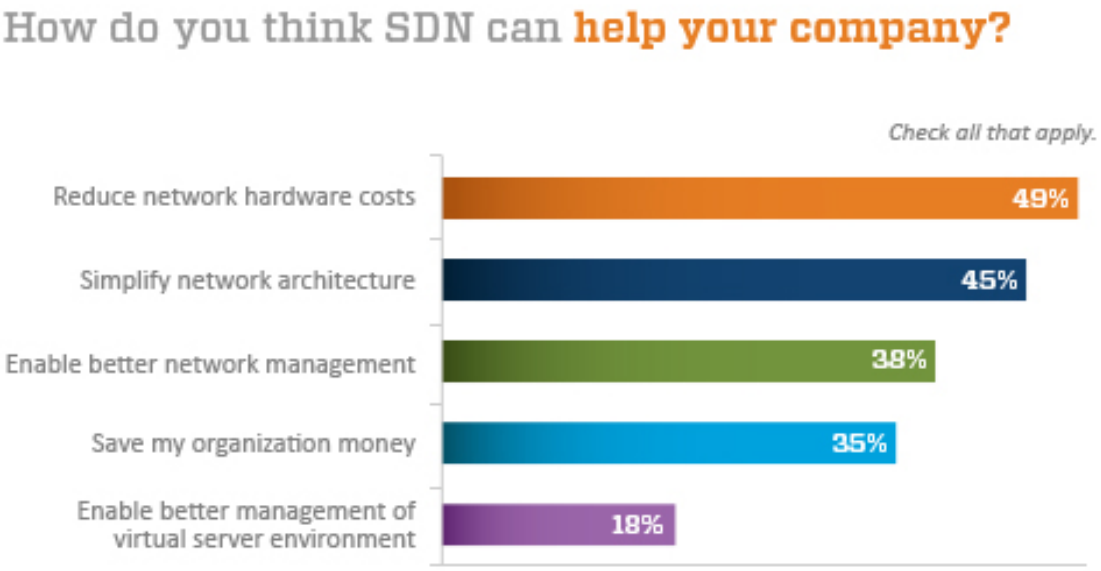


Tabla 1.1: Ahorro costes SDN

1.4. Estructura del documento

Siguiendo al capitulo de Introducción, el contenido del proyecto se completa con 7 capítulos adicionales, los cuales se resumen a continuación para facilitar su lectura:

- **Glosario**
- **Capítulo 1: Introducción** Breve introducción del proyecto en el que se expone la motivación del proyecto, los objetivos a alcanzar, el entorno socio-económico que lo enmarca y la estructura del documento.
- **Capítulo 2: Estado del arte** Capítulo dedicado al análisis del estado del arte de las tecnologías usadas para el desarrollo del proyecto y el estado del marco regulador actual.
- **Capítulo 3: Desarrollo de la solución técnica** Visión general de la solución del sistema llevado a cabo. Se enumeran los requisitos de las capacidades, funcionalidades y restricciones de las tecnologías aplicadas. Se describe el despliegue y las características de la solución para poder crear diferentes escenarios de red. Además se plantean diferentes alternativas y cual fué la escogida para la resolución del proyecto.

- **Capítulo 4: Diseño de escenarios y evaluación de resultados** Se describen diferentes escenarios de red para probar el funcionamiento de SFC. Se proporcionan scripts de despliegue y se evalúa el correcto funcionamiento de la solución.
- **Capítulo 5: Planificación del trabajo y presupuesto del proyecto** Planificación detallada del proyecto y presupuesto final del mismo
- **Capítulo 6: Conclusión y trabajos futuros** Conclusiones a las que se han llegado tras la finalización del trabajo y posibles mejoras futuras.
- **Capítulo 7: Anexos** Se presentan dos anexos para complementar la solución de trabajo.
- **Bibliografía**

Capítulo 2

Estado del arte

En este capítulo se describirá la base teórica sobre la que se soporta el proyecto, explicando que es el Cloud Computing, OpenStack y sus diferentes módulos, que es Google Cloud, OpenvSwitch y Service Function Chaining.

2.1. Cloud Computing

El termino Cloud Computing hace referencia a una nueva visión tecnológica y un nuevo modelo de negocio en el que se permite el acceso a servicios de almacenamiento, acceso y uso de recursos informáticos alojados en la red. Toda la gestión del hardware es gestionada por el proveedor del servicio Cloud garantizando cierta disponibilidad, potencia y recursos bajo unos niveles de servicio determinados (SLAs).

De esta forma, el cliente o usuario puede obviar la parte correspondiente al hardware y enfocarse en la actividad central de su negocio (e.g. aplicaciones web, red privada empresarial, almacenamiento de bases de datos). Por otra parte un usuario final puede disponer de un dispositivo de altas prestaciones para tareas que requieran un sistema con muchos recursos gracias a la computación en la nube.

La figura 2.1 muestra el paradigma del Cloud Computing.

Figura 2.1: Cloud Computing

De acuerdo con el NIST [3], para que un servicio pueda considerarse Cloud, debe cumplir cinco condiciones:

- **Servicio bajo demanda:** Un cliente puede aprovisionarse de recursos de computación, tales como almacenamiento en red o tiempo de cómputo automáticamente sin la intervención humana del proveedor de servicios.
- **Acceso amplio desde la red:** Todas las capacidades de red deben ser accesibles a traves de mecanismos y plataformas estándar (e.g móviles, navegadores web, tablets).
- **Puesta en común de recursos:** Los recursos Cloud deben tener una puesta en común para servir a varios clientes a través de un modelo *multi-tenant*[1]. De esta manera los recursos físicos y virtuales se asignarán o reasignarán dinámicamente para satisfacer la demanda de consumo.
- **Rapida elasticidad:** Las características de los recursos contratados deben ser dinámicas. Es decir, los recursos asignados deben aumentar o disminuir bajo demanda del usuario de forma automática o con la mayor celeridad posible.
- **Capacidad de medición de servicio:** Los sistemas Clouds deben ser controlados y optimizados a través de métricas que permitan un nivel de abstracción apropiado al tipo de servicio (e.g. almacenamiento, ancho de banda, ...). Los recursos deberán ser monitorizados, controlados y bien presentados para ofrecer transparencia al proveedor y al cliente.

2.1.1. Tipos de sistemas Cloud

Una estructura de Cloud Computing puede clasificarse en base a dos criterios:

- Según el modelo de servicio, existen 3 clasificaciones [5], como se muestra en la figura 2.2[2]
  - **Software as a Service (SaaS):** El cliente tiene la capacidad de usar una aplicación que corre sobre una *infraestructura Cloud* [3]. Las aplicaciones son accesibles desde desde varios dispositivos a través de un cliente web o una interfaz de programa. El cliente no puede controlar la infraestructura subyacente incluyendo red, servidores, sistemas operativos o almacenamiento.
  - **Platform as a Service (PaaS):** El cliente tiene la capacidad de desplegar sobre una infraestructura Cloud, una aplicación creada a través de lenguajes de programación, librerías, servicios y herramientas proporcionadas por el proveedor. El cliente no tiene control sobre la infraestructura subyacente pero si tiene control sobre las aplicaciones desplegadas y la configuración de estas.

[1] En el contexto de Cloud Computing el término Tenant se entiende como un usuario o grupo de usuarios que comparten acceso común a una instancia con privilegios específicos.

[2] https://teuno.com/servicios-en-la-nube-saas-iass-paas/

[3] Una infraestructura Cloud es el conjunto de hardware y software que permiten las cinco características esenciales del Cloud Computing.

  - **Infrastructure as a Service (IaaS):** El cliente tiene la capacidad de disponer de recursos de computo, almacenamiento, red y otros servicios fundamentales para dar la posibilidad al usuario de desplegar y correr software arbitrario, que puede incluir sistemas operativos y aplicaciones. El cliente no tiene control sobre la infraestructura subyacente, pero tiene control sobre el sistema operativo, almacenamiento, aplicaciones desplegadas y la posibilidad de configuración de algunos componentes de red tales como el firewall.

Figura 2.2: Tipos de Cloud según el servicio

- Según el modelo de despliegue, existen 4 clasificaciones, como se muestra en la figura 2.3[4]

  - **Nube privada:** Los servicios y la infraestructura se encuentran dentro de una organización y son de uso exclusivo de esta. La organización debe ser el propietario y debe ser administrarla por ésta, terceros o una combinación de los dos.
  - **Nube comunitaria:** Nube desplegada para el uso exclusivo de una comunidad que normalmente comparten objetivos o asuntos (e.g policía, salud). Una o mas de las organizaciones de la comunidad debe ser el propietario y debe administrarla, ceder la administración a terceros o hacerlo entre ambos.
  - **Nube pública:** Nube desplegada en servidores externos a la empresa o usuario que desea hacer uso de ella. Es un modelo de negocio donde el proveedor Cloud factura según el tiempo y los recursos contratados. Permite una gran escalabilidad y una reducción de inversión en infraestructura al cliente que la contrata.

[4]http://cloudcomputinguigv.blogspot.com.es/2015/11/tipos-de-nubes.html

– **Nube híbrida:** Nube que combina las características de dos o mas tipos de nube diferentes (privada, publica comunitaria) para realizar diferentes funciones dentro de una misma organización. Según el tipo de operación se usara una u otra, por ejemplo, una empresa puede alojar su website de información en una nube pública donde le es más rentable y no es un factor clave la seguridad, y su website de comercio electrónico en una nube privada donde es seguro y escalable.

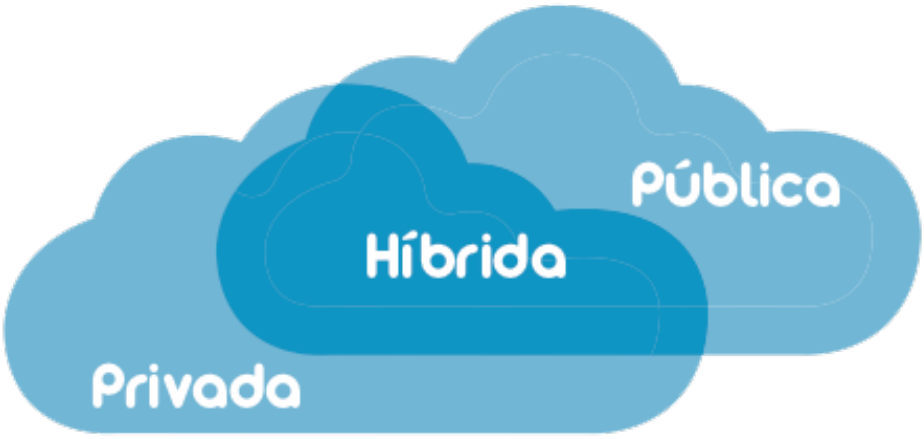


Figura 2.3: Tipos de Cloud según el despliegue

2.2. OpenStack

OpenStack[5] es un proyecto Open Source de computación en la nube, que permite controlar recursos de computación, almacenamiento y redes a través de un dashboard y una serie de *APIs*. [6]

La arquitectura es *distribuida* [7], utiliza varios nodos especializados en distintos servicios que en su conjunto y correctamente conectados mediante redes de gestión y datos, forman una estructura Cloud completa, encuadrándose en la definición de IaaS.

En la figura 2.4[8] se muestra un despliegue sobre tres nodos

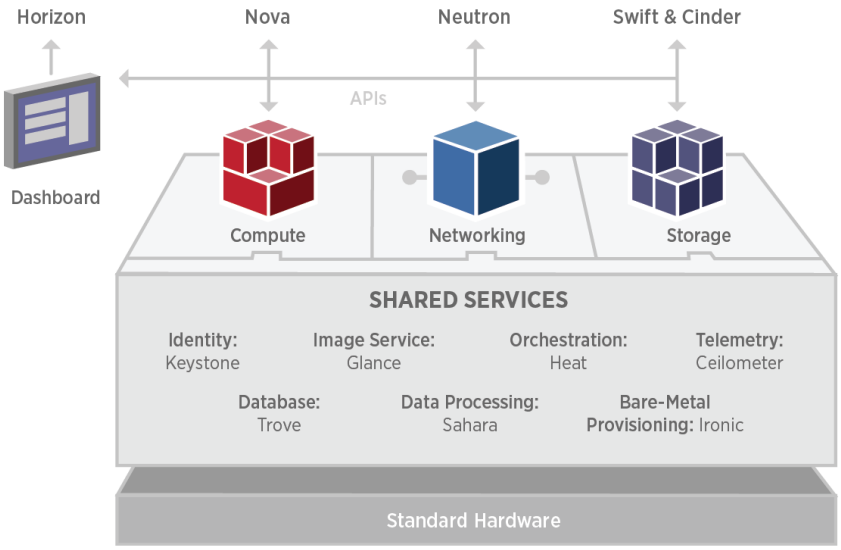


Figura 2.4: Nodos de OpenStack

2.2.1. Arquitectura básica de OpenStack

OpenStack [1] está dividido en diferentes nodos con distintas funcionalidades que trabajan en conjunto. La integración de todos ellos es posible gracias a las interfaces de programación de aplicaciones (API), por lo tanto, es posible reemplazar un servicio por otro mas actualizado o con otras funcionalidades que se adapten mejor al tipo de proyecto, respetando la forma de comunicación mediante APIs usada.

[5] Todaas las figuras de las arquitecturas de los diferentes módulos provienen de https://docs.openstack.org/arch-design/design.html

[6] API (Application Programming Interface) es un conjunto de subrutinas, funciones y métodos que ofrece cierta biblioteca para ser utilizado por otro software como capa de abstracción

[7] Para nuestro despliegue, únicamente usaremos un nodo, lo cual es perfectamente funcional y exactamente igual a un despliegue distribuido para nuestros casos de uso

[8] http://www.unixarena.com/2015/08/openstack-architecture-and-components-overview.html

No es necesario implementar todos los nodos desde el momento inicial, el usuario puede elegir cuales son los que va a necesitar el diseño de su entorno Cloud inicial y cuales implementar en un futuro.

En la figura 2.5 se puede ver un despliegue completo con todos los servicios y sus diferentes interacciones.

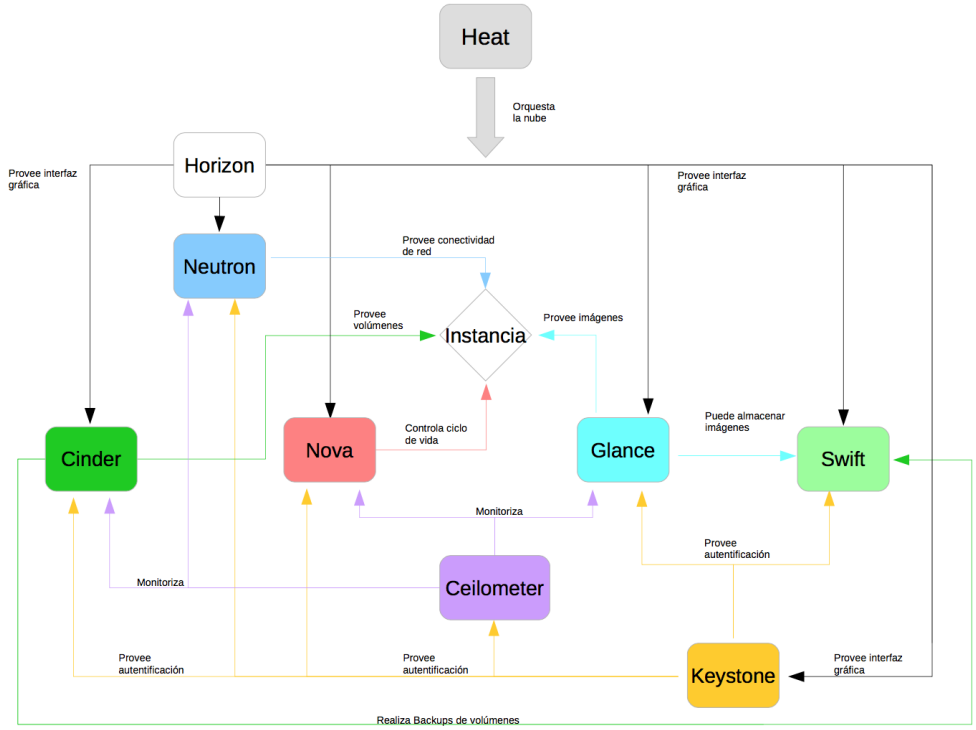


Figura 2.5: Estructura de OpenStack

2.2.1.1. Servicios principales de OpenStack

OpenStack nos permite alojar los servicios en diferentes nodos, o compartir el alojamiento entre todos, según decida el arquitecto Cloud. Un servicio o componente está compuesto por varios subservicios tales como *q-dhcp o nova-compute* [9] que podrían estar en diferentes nodos, pero en este proyecto se alojaran todos en el mismo.

No se van a instalar todos ellos, ya que requieren de grandes recursos hardware y no aportarían nada a los escenarios de red que se desplegarán para probar SFC.

A continuación se detallan las principales funcionalidades de cada servicio.

[9] A medida que se actualizan las versiones de OpenStack, estos servicios o subservicios pueden cambiar de nombre, fusionarse o desaparecer

2.2.1.2. Horizon

Horizon [6], figura 2.6, provee un dashboard desde el cual se pueden controlar el resto de servicios para el administrador o el usuario final.

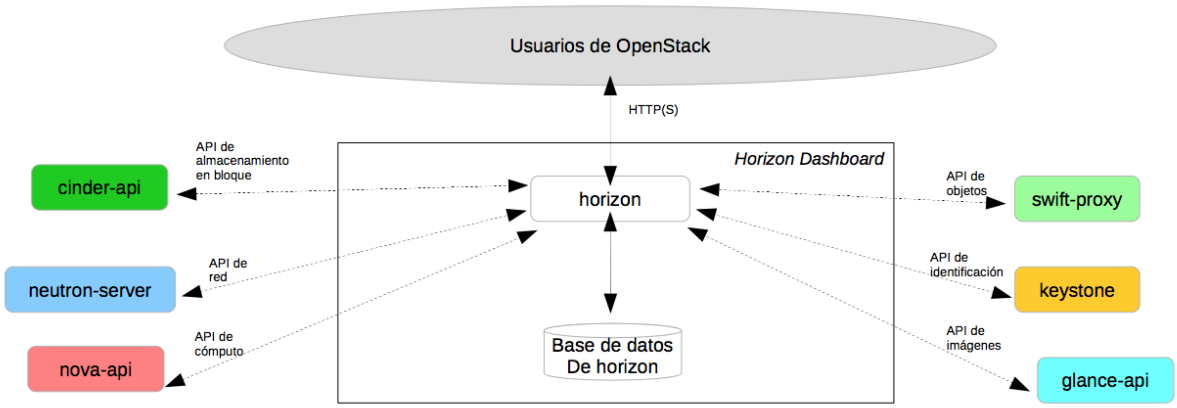


Figura 2.6: Estructura de Horizon

Características: [2]

- Utiliza las APIs para establecer comunicación con los otros servicios.
- Aplicación web desarrollada con *Django* [10].
- Es personalizable para cambiar la apariencia y la funcionalidad.
- Dispone de una base de datos SQLite3.

[10] Django es un framework escrito en Python para el desarrollo web

2.2.1.3. Nova

Nova Compute [7], figura 2.7, es el servicio encargado de levantar máquinas virtuales, tiene le propiedad de poder iniciar, parar, suspender o reiniciar las instancias, puede desplegarse sobre varios nodos según la carga de trabajo que necesite el entorno Cloud.

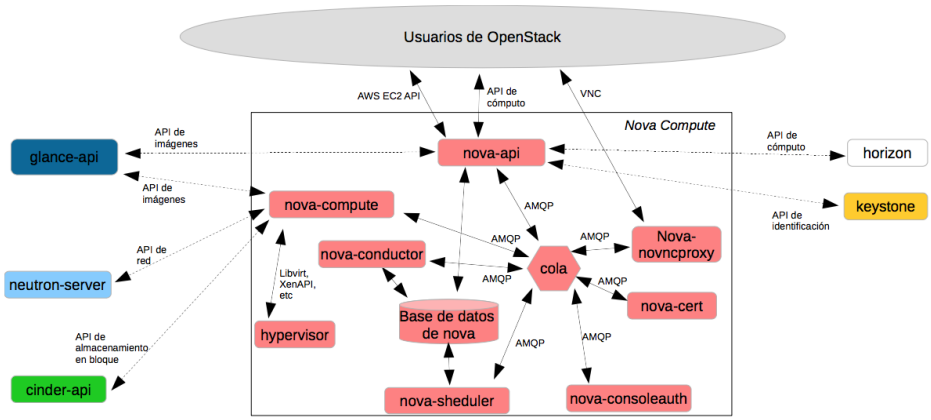


Figura 2.7: Estructura de Nova

Características de los componentes:

- **Nova-api** es el responsable de dar una respuesta a las peticiones del usuario a traves del API de Amazon EC2, OpenStack Compute o del dashboard Horizon. Se comunica con KeyStone para proporcionar autenticación y con glance-api para comprobar las imágenes disponibles que usan las instancias.

- **Nova-compute** se encarga de la creación y finalización de las instancias de VMs mediante el uso de la API del *hipervisor*[11] que se esté utilizando. Nova-compute se comunica con neutron-server para proporcionar a las instancias conexión con las redes virtuales y a su vez con cinder-api, para poder montar volúmenes en dichas instancias.

- **Nova-schedule** se encarga de la planificación de los recursos de computo, toma las peticiones de instancias desde la cola y determina donde deben crearse, por último actualiza su estado en la base de datos.

- **Nova-conductor** se encarga de hacer posible la interacción entre nova-compute y la base de datos.

- **Nova-consoleauth** autoriza a los usuarios que deseen acceder a los proxis de consola. Estos proxis pueden ser nova-novncproxy, nova-xvpnvncproxi o nova-spicehtml5proxy.

[11] Un hipervisor es una plataforma de software la cual, tiene la capacidad de utilizar diferentes sistemas operativos sobre una misma máquina a través de diversas técnicas de control. Por defecto, en este proyecto se utilizará la API libvirt correspondiente al hipervisor QEMU.

- **Base de datos** es donde se almacenan los estados de las instancias que se están ejecutando, el tipo de instancias disponibles, proyectos disponibles y redes creadas.

- **Nova-cert** proporciona certificados X509 para permitir el acceso a las instancias.

2.2.1.4. Keystone

Keystone [8], figura 2.8, se encarga de la autenticación de los diferentes servicios de OpenStack así como de los usuarios y proyectos.

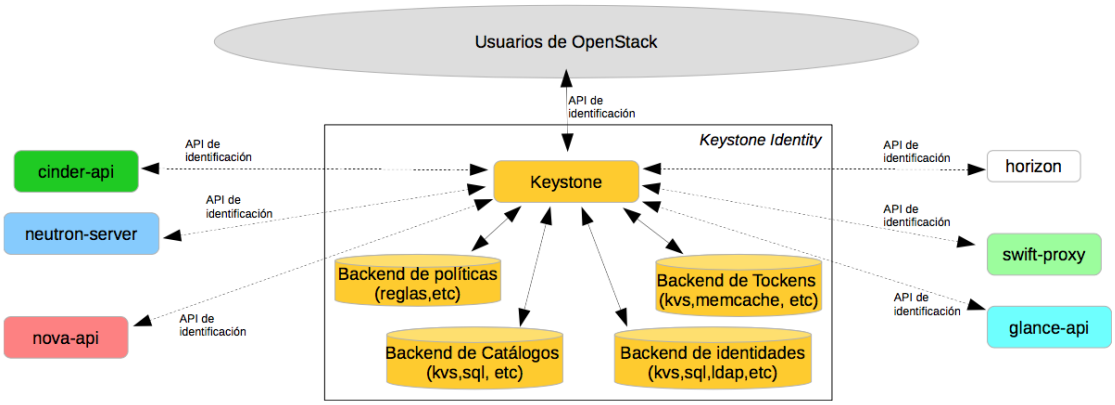


Figura 2.8: Estructura de Keystone

Las principales características de Keystone son:

- Gestiona los usuarios, tenants, roles y a que proyectos pertenecen cada uno.

- Keystone se encarga de autenticar al resto de servicios del entorno Cloud respondiendo las peticiones que llegan a su API.

- Proporciona una lista de servicios disponibles a los usuarios y sus correspondientes API end-point.

Para entender las características descritas anteriormente es necesario detallar algunos conceptos

- **Usuario:** es la representación digital de quien usa OpenStack. Un usuario valida sus peticiones en KeyStone a través de sus credenciales. Los usuarios pueden tener tokens asignados para acceder a determinados recursos y restringir otros.

- **Token:** es una línea de caracteres alfanuméricos usados para el acceso a las APIs y los diferentes recursos de OpenStack. Los tokens tienen una vida limitada.

- **Tenant:** es un usuario o grupo de estos que comparten acceso común a una instancia con privilegios específicos.
- **API end-point:** dirección URL donde se encuentra la API del servicio.
- **Rol:** cada usuario tienen un rol asignado, este rol define los privilegios que tiene sobre un tenant asociado. Un usuario puede tener el rol de administrador en un tenant, y por otro lado el rol de usuario sin privilegios en otro.

2.2.1.5. Neutron

Neutron Network [9], figura 2.9, se encarga de proporcionar conectividad de red para las interfaces virtuales (vNICs) creadas por Nova. Es el responsable de la gestión de las redes virtuales entre las instancias y la conexión de éstas con redes externas.

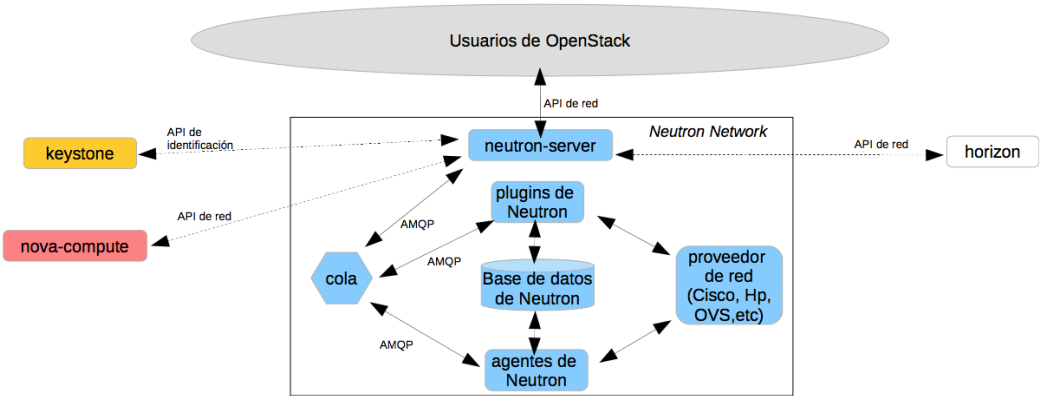


Figura 2.9: Estructura de Neutron

Sus características principales son:

- Neutron controla todo los aspectos del acceso a la infraestructura virtual de red (Virtual Networking Infrastructure) y solo el acceso a la infraestructura física de red (Physical Networking Infrastructure) en un entorno OpenStack.
- Los plugins y agentes son los que se encargan de procesos de red tales como el asignamiento de direcciones IP, creación de redes o asignamiento de puertos.
- Se pueden añadir funcionalidades avanzadas mediante el uso de otros plugins y agentes, como por ejemplo firewalls, SFC o VPNs.
- Se pueden asignar direcciones IP de redes externas a redes internas.
- Existen grupos de seguridad para controlar las conexiones entrantes y salientes a los puertos de cada instancia, a modo de firewall. Una instancia puede estar asociada a más de un grupo de seguridad.

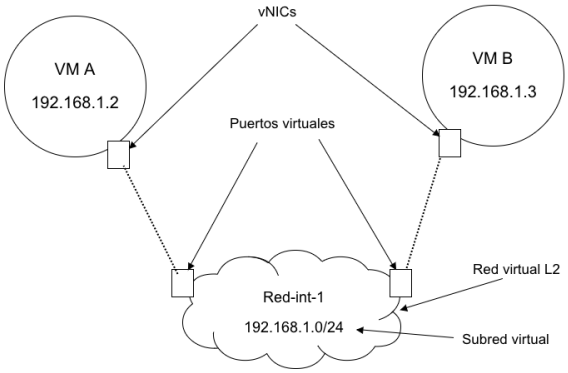


Figura 2.10: Asociación vNIC-Puerto virtual

- En la figura 2.10 se puede ver la asociación de una vNIC de una instancia a un puerto de una red virtual.

  1 El tenant crea una red Red-int-1.
  2 El tenant asocia a dicha red una subred (192.168.1.0/24).
  3 El tenant levanta la instancia A asociando una vNIC a la subred Red-int-1.
  4 Nova indica a Neutron que cree un nuevo puerto virtual en la red Red-int-1.
  5 Neutron se encarga de crear el puerto virtual, y típicamente mediante el agente DHCP se le asigna una dirección IP.

2.2.1.6. Swift

Swift Object Storage [10], figura 2.11, ofrece almacenamiento y recuperación de objetos con una simple API.

Es necesario entender una serie de conceptos que permiten a los objetos y su sistema:

- El sistema de objetos se organiza por Cuentas, Contenedores y Objetos.
- Una cuenta define un espacio de nombres para uno o varios contenedores y representa el máximo nivel de jerarquía. En el sistema OpenStack, una cuenta equivale a un proyecto o usuario.
- Un contenedor es el encargado de almacenar y definir un espacio de nombre para objetos. Pueden existir varios objetos con el mismo nombre en diferentes contenedores.
- Un objeto representa a la entidad que almacena información tal como documentos, imágenes o cualquier tipo de archivo. Tiene un identificador único que permite poder recuperar el objeto independientemente de donde se encuentre físicamente.

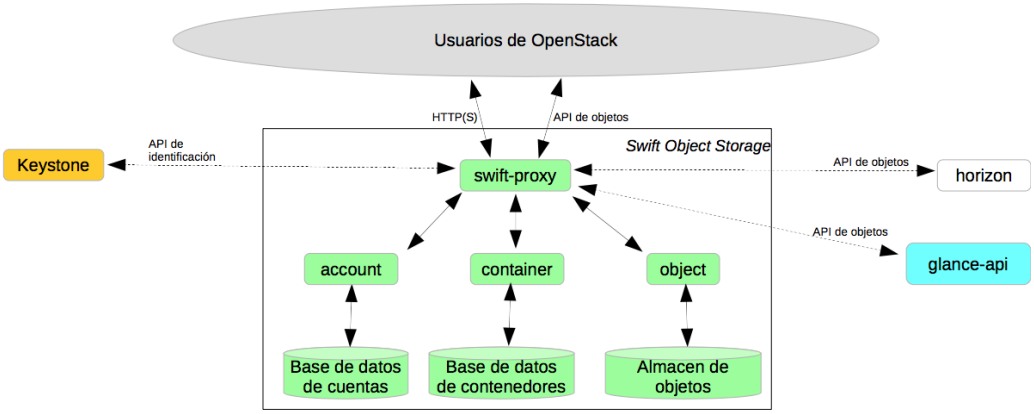


Figura 2.11: Arquitectura de Swift

Las principales características son:

- Account-server se encarga de administrar las cuentas de Swift.
- Container-server se encarga de administrar los contenedores de Swift, estos contenedores de objeto son típicamente conocidos como carpetas de objetos.
- Object-server se encarga de administrar los objetos de Swift.
- Swift-proxy está a la espera de peticiones de la API REST proporcionada a los clientes y de la API de objetos del sistema OpenStack. Las posibles peticiones pueden ser creación de contenedores, modificación de meta-datos o la subida de archivos. Swift-proxy se comunica con KeyStone para lograr autenticación.

2.2.1.7. Cinder

Cinder [11], figura 2.12, es el servicio de almacenamiento de en bloque de OpenStack.

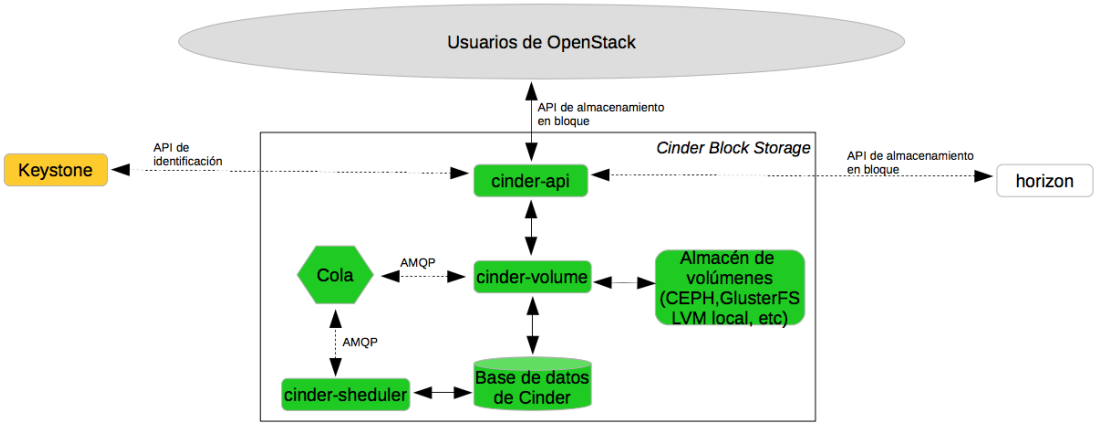


Figura 2.12: Arquitectura de Cinder

Almacenamiento en bloque es lo mismo que referirse a almacenamiento de volúmenes. Estos volúmenes están vinculados a instancias que se están ejecutando.

Los volúmenes se pueden vincular a una instancia, seguidamente desvincularlos y todos los datos se mantendrían intactos, es decir, son persistentes.

Las principales características de Cinder son:

- Cinder-api se encarga de las peticiones del API y las traslada hacia cinder-volume para que sean procesadas. También se encarga de la autenticación a traves de Keystone.
- El API de almacenamiento en bloque tiene la capacidad de manipular volúmenes, hacer snapshots y elegir que tipo de volúmen se desea.
- Cinder-volume se encarga de interactuar con la base de datos para controlar el estado de los volúmenes. Interactúa con cinder-scheduler usando la cola de mensajes.
- Cinder-scheduler selecciona el tipo de proveedor de almacenamiento para crear volúmenes.
- En la base de datos se almacena el estado de los volúmenes.

2.2.1.8. Glance

Glance Image Storage [12], figura 2.13, se encarga de proporcionar una colección de imágenes disponibles para ser usadas por las instancias.

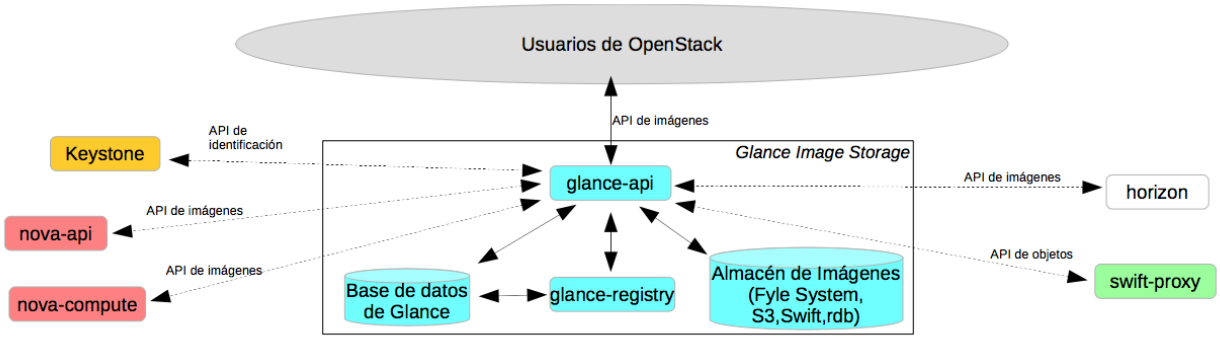


Figura 2.13: Arquitectura de Glance

Las imágenes son archivos únicos que guardan los contenidos y toda la estructura de un medio de almacenamiento.

Estas imágenes pueden ser utilizadas para la distribución de sistemas operativos, o snapshots de éstos. Una snapshot es una imagen que almacena el estado congelado del sistema operativo en un momento determinado.

Las principales características de Glance son:

- Glance-api está a la espera de peticiones para poder almacenar, listar y ofrecer imágenes.
- Glance-registry se encarga de almacenar, recuperar y procesar los meta-datos de las imágenes.
- La base de datos almacena los meta-datos de las imágenes.

2.2.1.9. Heat

Heat [13] es el orquestador de stacks de OpenStack. Estos stacks pueden contener servicios de distintos proveedores de Cloud. A través de plantillas de orquestación es posible el despliegue de escenarios de manera automatizada.

Sus principales características son:

- Los stacks son plantillas del tipo AWS CloudFormation de Amazon o formato nativo HOT (Heat Orchestration Template).
- Permite orquestar diferentes recursos y todas sus dependencias.

2.2.1.10. Ceilometer

Ceilometer [14] es el servicio que se encarga de monitorizar todas las métricas del entorno OpenStack para poder proporcionar datos de facturación, estadísticos o de escalabilidad.

Sus principales características son:

- Mediante la asignación de agentes a los diferentes servicios de OpenStack, se consigue monitorizar: congestión de la red, carga de cpu, almacenamiento, etc. Al mismo tiempo, un colector se encarga de realizar sondeos a los agentes de medición para recopilar las estadísticas actualizadas. Si ciertas estadísticas superan un umbral ya definido, se dispara una alarma hacia el agente que la provoca, el cual dirigirá esta alarma hacia un agente de alarmas.
- Los clientes pueden hacer peticiones mediante el API al colector para visualizar las estadísticas.

2.3. Software Defined Networking

Las redes definidas por software [15], o más comúnmente conocidas como Softare Defined Networking (SDN), tienen el propósito de proveer una serie de interfaces abiertas para poder controlar la conectividad que proporcionan los recursos de la red y el tráfico entre ellos, a través del desarrollo de software.

Todo esto se consigue gracias a la separación del plano de control del plano de datos. Esta migración del control, permite a la infraestructura subyacente, poder ser abstraída para así hacer posible que las aplicaciones y servicios de red puedan tratar a la red como una entidad virtual, de tal manera, que se pueden conseguir redes altamente escalables y flexibles que se ajusten a las necesidades necesarias de cada entorno.

2.3.1. Arquitectura SDN

Para diseñar la arquitectura de las redes definidas por software se postulan tres premisas o principios básicos, para así poder solventar las desventajas de las redes tradicionales:

- Separación del plano de control y datos, que posibilita el control sobre los diferentes recursos de la red. Este control se lleva a cabo a través de la interfaz de plano de control (CPI), que hace de intermediario entre los elementos de la red y el controlador SDN, permitiendo al controlador la gestión de funcionalidades sobre la red y los elementos de ésta.
- El estado y los recursos abstractos de la red deben poder ser expuestos a aplicaciones externas.
- Debe existir un control centralizado, que permita tener una amplia visión de los recursos de la red y el despliegue de los mismos.

La arquitectura SDN, figura 2.14[12], está compuesta por tres capas y dos APIs para acceder a las capas de datos y control.

[12] http://www.openairinterface.org/?page_id=466

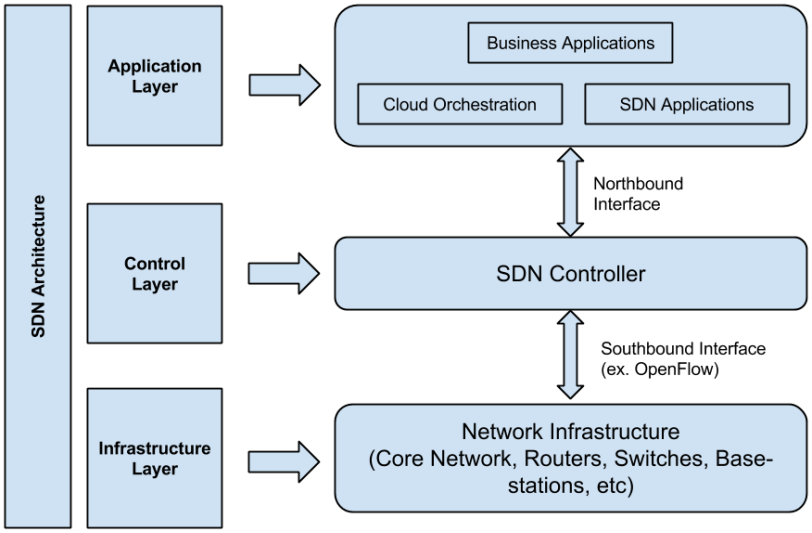


Figura 2.14: Arquitectura de SDN

En la capa de plano de datos se encuentran los elementos de red, que pueden ser o bien, dispositivos tradicionales o dispositivos programables. Entre los programables se encuentran por ejemplo los switches OpenFlow, que se encargan de el reenvío de paquetes de la red según las reglas que recibe de el controlador SDN a través de las interfaces south-bound. Las APIs south-bound son las encargadas de hacer llegar las peticiones del controlador hacia los switches OpenFlow y de la comunicación entre ellos.

En la capa central se encuentra el plano de control, donde pueden coexistir varios controladores, cada uno manteniendo el estado de un fragmento de la red. En función de la extensión de la red, los controladores interpretan los requerimientos de las aplicaciones para así poder ejercer un control más específico sobre los recursos de la red.

Los controladores se comunican con las aplicaciones SDN a través de las interfaces north-bound, que se encargan de interpretar los requerimientos de las aplicaciones y las políticas de red, para así poder comunicar al controlador que servicios debe habilitar. Estos servicios pueden ser encaminamiento de red, seguridad, eficiencia energética, ingeniería de tráfico, seguridad y otro tipo de servicios de gestión de red.

En la capa de aplicación residen las aplicaciones SDN, que comunican sus requisitos de red al plano de control a través de las interfaces north-bound.

En esta capa se alojan los orquestadores de red, y en el caso de este proyecto, es donde se aloja OpenStack.

2.3.2. OpenFlow

OpenFlow [16] es una interfaz de comunicación estandarizada entre los planos de datos y de control en una arquitectura del tipo SDN. Gracias a OpenFlow se consigue el control sobre los dispositivos de red situados en la capa de datos, tanto virtuales como físicos. En la figura 2.15[13] se observan los componentes de un switch OpenFlow y sus interacciones.

Al conjunto de tablas de flujos y el secure channel, se le denomina Switch OpenFlow. Este canal de seguridad se encarga de comunicar el switch con el controlador a través del protocolo OpenFlow, mediante el cual se especifican las entradas en la tabla de flujos.

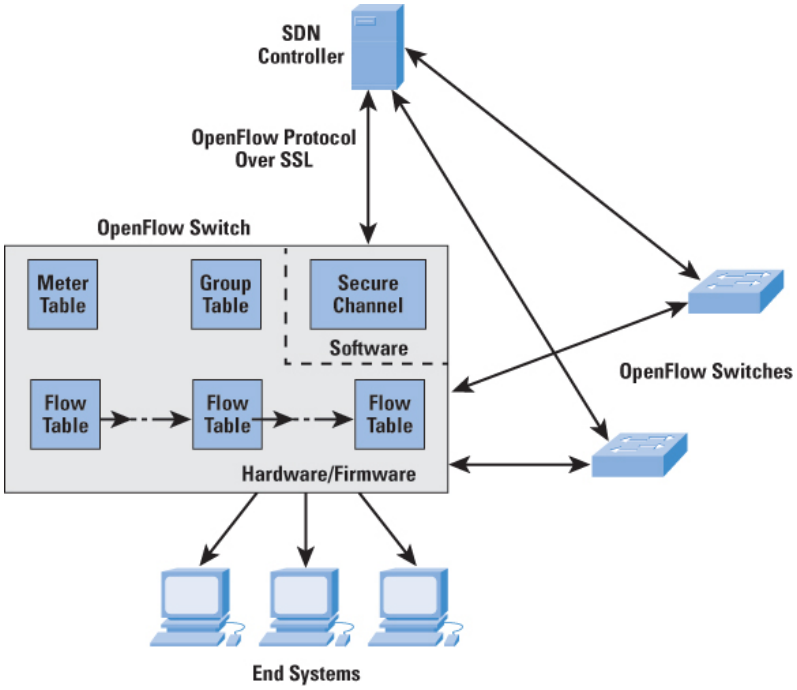


Figura 2.15: Switch OpenFlow

Para identificar el tráfico de red, OpenFlow hace uso del concepto de flujo, el cual se basa en reglas de correspondencia ya definidas, estas reglas también pueden ser programadas por el controlador SDN.

Dentro de un switch OpenFlow, como se ve en la figura 2.16[14], pueden existir una o mas tablas de flujo, mediante las cuales se establecen reglas de coincidencia o matching para los paquetes del tráfico entrante. Estas tablas de flujo constan de tres campos:

- Un campo de cabecera que define el tipo de flujo.
- Un campo de contadores para contabilizar el número de coincidencias de paquetes.

[13] https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html
[14] https://blog.zhaw.ch/icclab/a-learning-switch-in-openflow-1-3/

- Las acciones que se deben realizar cuando coinciden un paquete con una tabla de flujos.

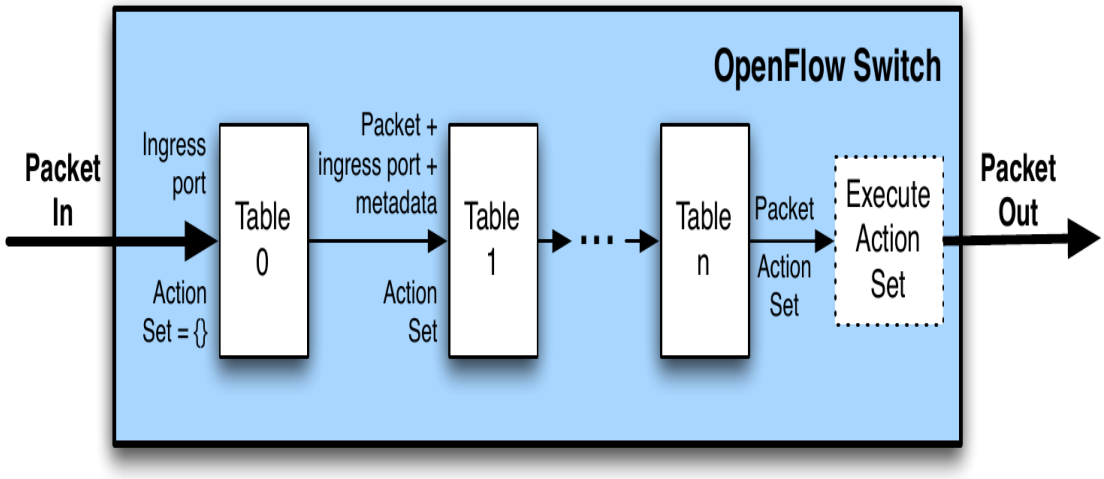


Figura 2.16: Acciones de OpenFlow

A la entrada de un paquete en el switch, se determina mediante la primera tabla las coincidencias con el paquete, y si las hay, se copiará la acción correspondiente en una cola de acciones, seguidamente, se vuelve a comparar el paquete con la siguiente tabla y así con todas las que existan. Una vez finalizadas las comparaciones, se ejecutarán todas las acciones correspondientes en el Execute Action Set en el orden que proceda. Si el switch no encontrara coincidencias con un paquete entrante, éste sería devuelto al controlador.

Todos los switches con el protocolo OpenFlow deben soportar como mínimo tres tipos de acciones para sus tablas de flujo:

- Enrutar los paquetes entrantes a través de un puerto del switch.
- Eliminar paquetes entrantes.
- Reenviar un paquete entrante de nuevo al controlador, para que así éste pueda decidir si crear una nueva tabla de flujo con su correspondiente acción o procesar el paquete de una manera diferente.

Los campos que se utilizan para realizar las reglas de matching en las primeras versiones de OpenFlow son los siguientes [22]

- Direcciones MAC origen y destino
- Direcciones IPv4 origen y destino
- Puerto de entrada al switch
- Protocolo IPv4

- VLAN ID
- Puertos origen y destino
- Ethertype
- ToS IPv4
- Prioridad VLAN

2.3.3. Protocolo OpenFlow

Para comunicar el switch OpenFlow con el controlador, se establece el protocolo OpenFlow a través del secure channel perteneciente al switch. En la figura 2.17[15], se inicia una sesión TCP entre el switch y el controlador por parte del switch a través de una dirección IP y un puerto determinado. Para que la conexión se establecida correctamente, se deben realizar dos negociaciones, primero se debe establecer la versión del protocolo a utilizar, y seguidamente el descubrimiento de las capacidades del switch por parte del controlador.

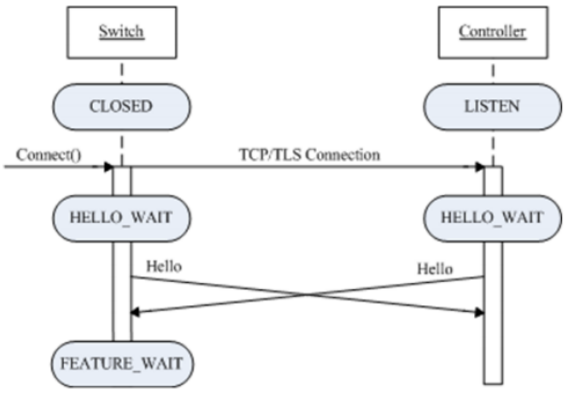


Figura 2.17: Secuencia OpenFlow

Existen tres tipos de mensajes:

- Controlador al switch: El controlador desea conocer y tratar las características y el estado del switch.
- Mensajes asíncronos: El switch interacciona con el controlador a la llegada de paquetes, cambio de estado o error.
- Mensajes simétricos: Los mensajes pueden ser enviados en cualquier dirección y son del tipo Hello, echo o vendor message.

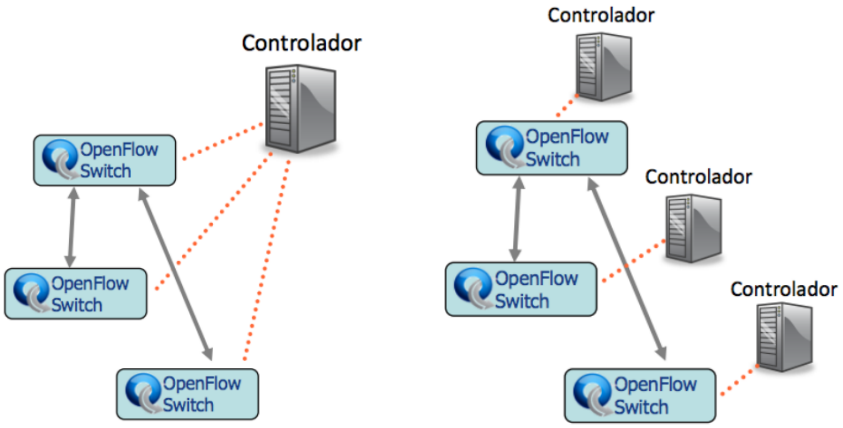
[15] http://flowgrammable.org/sdn/openflow/state-machine/

2.3.4. Controlador

La arquitectura SDN se puede establecer que gira en torno a su dispositivo principal, el controlador. Este dispositivo es el encargado de implementar las reglas de red y ejecutar las acciones que le indican las aplicaciones y a su vez distribuirlas a los dispositivos situados en la capa física.

El controlador es el dispositivo que se encuentra en el çentro"de la inteligencia de la red. Se encarga además de gestionar las entradas de flujo en los switch OpenFlow, y decide que hacer cuando un paquete no coincide con ninguna de esas tablas.

Existen varios tipos de configuraciones, como se observa en la figura 2.18[16], donde un único controlador se encarga de gestionar todos los switches, o cada switch es gestionado por su propio controlador.



(a) Arquitectura centralizada (b) Arquitectura distribuida

Figura 2.18: Arquitectura de controlador

Existen dos tipos de enrutamiento:

- Enrutamiento basado en flujo: Donde cada flujo individual es generado en el controlador, la tabla de flujo contiene únicamente una entrada por flujo.
- Agregado: Donde una entrada corresponde un grupo de flujos, la tabla de flujo contiene una entrada por cada categoría de flujos, estas entradas son configuradas a través de wildcards.

Dos tipos de comportamiento:

- Reactivo: El primer paquete del flujo ocasiona la creación de una entrada en la tabla de flujos, de esta manera se hace un uso más eficiente de ésta.
- Proactivo: Las tablas de flujo son completadas a priori por el controlador y así no se pierde tiempo en la creación de entradas cada vez que llega un flujo.

[16]https://www.slideshare.net/PallaviChhikara/sdn-ppt

2.4. Network Function Virtualization

Los servicios tradicionales de la industria de las telecomunicaciones eran llevados a cabo por los operadores de red. Cada servicio era provisto mediante la conexión de múltiples funciones ya desplegadas.

Los estrictos requisitos de encadenamiento, estándares de estabilidad y el incremento de servicios de corta duración, obliga a los proveedores de servicios de telecomunicación a actualizar constantemente sus equipos físicos.

2.4.1. NFV y VNF

NFV [17] fue introducido por el Instituto Europeo de Standards de Telecomunicaciones (ETSI) en 2012. NFV es un modelo de arquitectura de red que tiene como objetivo la virtualización de servicios de red. NFV propone que todas las funciones realizadas por nodos de red, deberían ser definidas en un modelo virtual, tal que todas las operaciones de red pudieran ser clasificadas dentro de bloques de construcción que pueden ser encadenados. Cada uno de estos bloques de construcción representaría un Virtualized Network Function (VNF).

Los VNFs proporcionan funcionalidades de red especificas, tales como encriptación/desencriptación, VPN, Balanceo de carga, Firewall, WAN Accelerator e Intrusion Detection System (IDS) basado en Deep Packet Inspection (DPI). La industria encargada en la especificación de NFV ha argumentado que la virtualización de servicios provee despliegues más rápidos que resultan en innovaciones a corto plazo. Otros objetivos de NFV incluyen la reducción del OPEX, reducción de uso de energía pudiendo apagar recursos sin utilizar y una mejor adaptación a los nuevos modelos de negocio.

En comparación con NFV, los dispositivos tradicionales requieren de mantenimiento, actualización de software subyacente, e instalaciones manuales. Un beneficio aparente de las VNFs es el tiempo de despliegue. El tiempo para levantar una máquina virtual y correr unas pocas lineas de código para ejecutar una VNF, no es comparable con la instalación de un dispositivo hardware tradicional. En la figura 2.19 se muestra una comparación entre una arquitectura tradicional y una NFV.

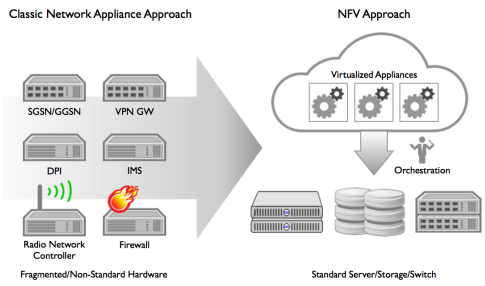


Figura 2.19: Comparación NFV-Arquitectura tradicional

2.4.2. Arquitectura NFV

Las VNFs se pueden desplegar para compartir los diferentes recursos físicos que ofrece la infraestructura, así como también es posible reasignar nuevos recursos a éstas. Esto hace posible el despliegue de nuevas funciones de red en poco tiempo.

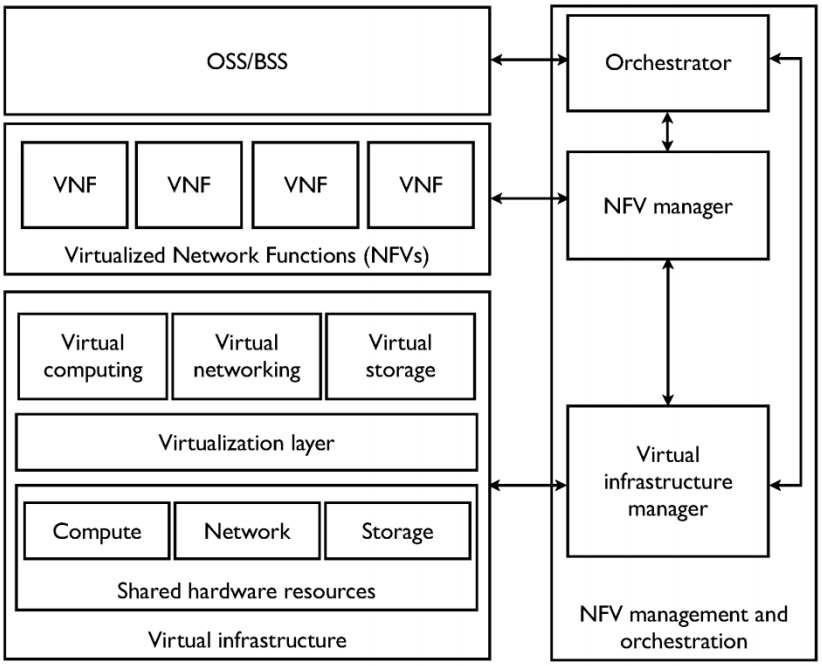


Figura 2.20: Arquitectura NFV

Existen tres componentes principales dentro de una arquitectura NFV (figura 2.20[17]): las funciones virtuales de red (VNFs), la infraestructura NFVI y la gestión y orquestación de NFV (NFV-MANO)

- VNFs: Son las funciones de red virtualizadas, se pueden instalar en maquinas virtuales. Una VNF es gestionada por un EMS, que se encarga de la creación, monitorización y rendimiento. El EMS proporciona información a los sistemas de soporte de operaciones (OSS). Estos OSS, unidos a los sistemas de apoyo a empresas (BSS) hacen posible a los proveedores a desplegar y gestionar soluciones extremo a extremo.

- NFVI: La infraestructua de un entorno NFV, se compone de todos los recursos de software y hardware que la hacen posible.

  Existe una capa de virtualización que abstrae los recursos físicos (computación, almacenamiento y conectividad). La arquitectura NFV puede ser variable y se aprovecha de la capa de virtualización existente, como puede ser un hipervisor que puede asignar los recursos necesarios a las VNFs.

- Orquestación y gestión NFV: La orquestación del entorno NFV está compuesto por un orquestador, además de gestores de infraestructura

[17] https://www.sdxcentral.com/nfv/definitions/nfv-elements-overview/

virtualizada (VIMs) y los gestores de las VNFs. En este bloque se lleva a cabo la gestión de los repositorios de datos, los puntos de referencias e interfaces que se utilizan para comunicar los diferentes componentes que forman la arquitectura, para así asegurar el correcto funcionamiento de las funciones virtuales en la NFVI.

El orquestador es el servicio encargado de gestionar y dirigir el servicio de red de un punto a otro que forman una cadena de VNFs. Por norma general solo existe un orquestador, sin embargo, pueden múltiples gestores de VNFs, que se encargan de las operaciones en el ciclo de vida de las funciones.

En el caso de que se quisiera desplegar un servicio de red donde aparezcan las funciones de firewall y sistema IDS, será NFV-MANO el encargado de decidir donde se sitúan estas funciones dentro de la red física. Estas VNFs a su vez serán controladas por los EMS y por NFV-MANO. La capa de virtualización será la encargada de proveer los recursos físicos en las ubicaciones que decidirá NFV-MANO a las funciones (VNFs).

2.5. Service Function Chaining

SFC [19] es la creación de un camino desde la fuente a el destino que cruza a través de unas funciones de red específicas, como pueden ser NATs, Deep Packet Inspections (DPIs) o firewalls.

En SFC existen dos tipos de servicios de encadenamiento, llamados unidireccionales o bidireccionales (figura 2.21). SFC unidireccional tiene la característica de que las cadenas de servicio de entrada y salida difieren entre sí. En SFC bidireccional las cadenas de servicio de entrada y salida hacen que los paquetes sigan las mismas funciones de red, pero en orden inverso.

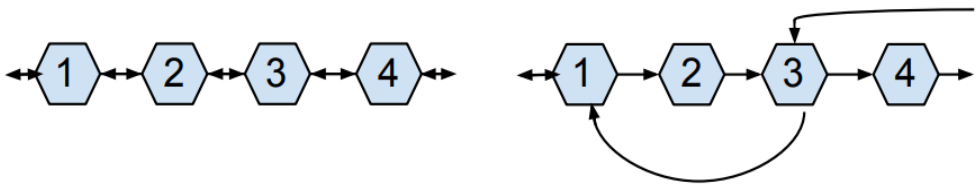


Figura 2.21: Bidireccional SFC (izquierda) vs. Unidireccional SFC (derecha)

Mediante el uso de SDN, las cadenas de servicio no solo pueden diferenciar por host o subred, sino también por proceso o aplicación. Estas diferenciaciones son posibles gracias a las reglas de OpenFlow, que pueden diferenciar paquetes por muchas más propiedades de lo que lo harían las reglas tradicionales de forwarding. Además, el hecho de que las reglas de OpenFlow pueden ser cambiadas rápidamente gracias al controlador SDN, da como resultado la posibilidad de preparar, alterar y deshabilitar las cadenas de servicio inmediatamente.

2.5.1. Definición de conceptos

Clasificación: Reglas de clasificación aplicadas al tráfico de red para la aplicación que las requiere. Las políticas de clasificación pueden ser específicas de cliente/red/servicio.

Clasificador: Elemento que aplica la clasificación.

Service Function Chain (SFC): Una Service Function Chain define un conjunto ordenado de Service Functions, junto con unas reglas de que deben ser aplicadas a los paquetes o flujos seleccionados, como resultado de la clasificación.

Service Function (SF): Una función es la responsable del tratamiento específico de los paquetes recibidos. Una Service Function puede actuar en varias capas de la pila de protocolo. Como un componente lógico, una SF se puede realizar como un elemento virtual, o ser integrado en un elemento de la red física. Una o mas SFs pueden ser implementadas en el mismo elemento de red.

Una SF debe ser consciente de la encapsulación SFC para así poder actuar correctamente según lo indicado en los paquetes, a su vez, también debe ser consciente de la no encapsulación SFC.

Service Function Forwarder (SFF): Un Service Function Forwarder es el responsable de reenviar el tráfico a una o mas SFs conectadas, de acuerdo con la información indicada en la encapsulación SFC, también como manejar el tráfico de vuelta de la SF. Además, un SFF es responsable de, entregar trafico al clasificador cuando sea necesario y soportado, transportar tráfico hacia otro SFF y terminar el Service Function Path (SFP).

Metadata: Proporciona la capacidad de intercambiar información entre clasificadores y SFs, también entre SFs.

Service Function Path (SFP): El Service Function Path es una especifiación de donde los paquetes a cierta SFP deben ir. Aunque puede ser muy concreto como para identificar las localizaciones exactas, también puede ser menos específico. El SFP proveé un nivel de abstracción entre la noción de service chain como una secuencia de SFs encadenadas, y la especificación completa de que SFF/SFs exactamente los paquetes tienen que visitar cuando atraviesen la red.

SFC Encapsulation: La encapsulación SFC provee, como poco, identificación del SFP, y es usada por las SFs y SFF que saben interpretar la encapsulación. La encapsulación SFC no es usada para el direccionamiento de paquetes. Además de identificación SFP, la encapsulación SFC lleva metadatos incluyendo información del plano de datos.

2.5.2. Arquitectura SFC en OpenStack

Todos los servicios de networking y las instancias de OpenStack Compute, se conectan a la red virtual mediante puertos, haciendo posible la creación de direccionamiento de tráfico a través de los servicios usando los puertos virtuales. Incluyendo estos puertos en una cadena de puertos, se habilita el direccionamiento de tráfico a traves de una o más instancias, que actúan como funciones de servicio. En la figura 2.22[18] se muestra una vista general de la arquitectura de SFC integrada en OpenStack.

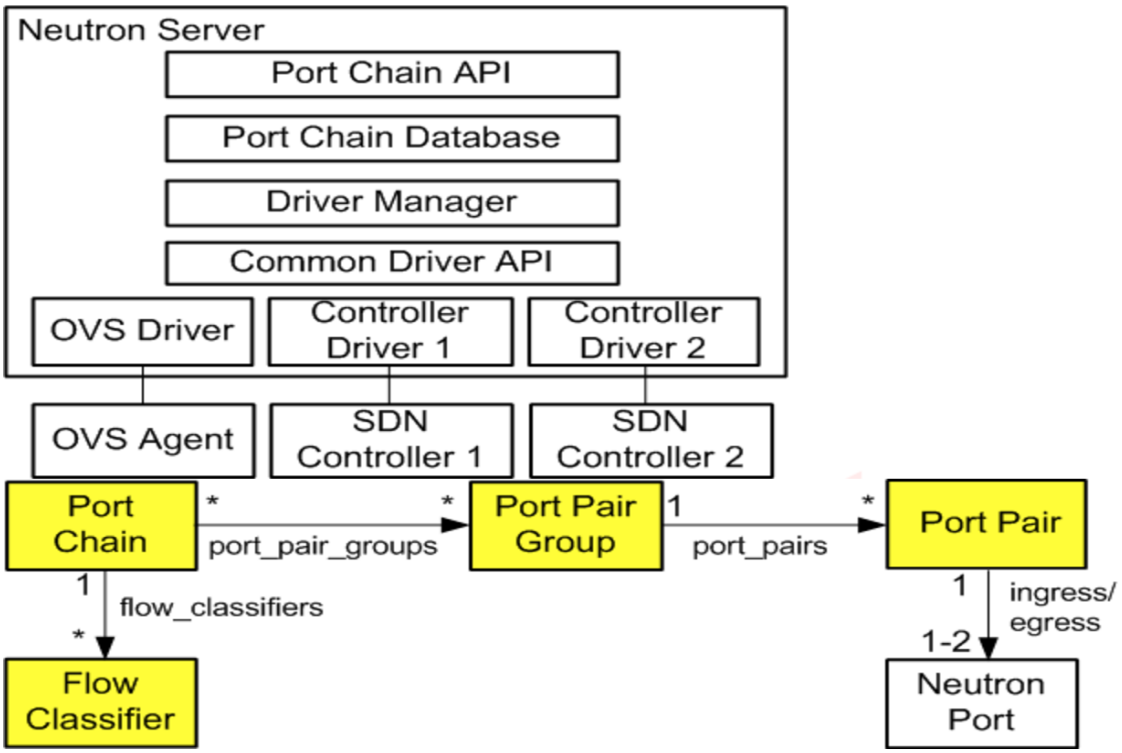


Figura 2.22: Arquitectura SFC

Una cadena de puertos, o camino de funciones de servicio (SFC), consiste en lo siguiente:

- Un conjunto de clasificadores de flujo que especifican los flujos de tráfico entrantes en la cadena
- Si una función de servicio involucra un par de puertos, el primer puerto actúa como el puerto de ingreso, mientras que el segundo actúa el puerto de salida. Si ambos puertos usan el mismo valor (ingress y egress), funcionan como un único puerto bidireccional.
- Una cadena de puertos (Port Chain) es una cadena de servicio unidireccional. El primer puerto actúa como la cabeza de la cadena de servicio, y el segundo

[18] https://docs.openstack.org/newton/networking-guide/config-sfc.html

puerto actúa como la cola. Una función de servicio bidireccional, consiste en dos cadenas de puertos unidireccionales.

- Un clasificador de flujo puede solo pertenecer a una cadena de puertos para prevenir la ambigüedad en cuanto a qué cadena debe manejar los paquetes de flujo. Sin embargo, se pueden asociar varios clasificadores de flujo con una cadena de puertos, ya que varios flujos pueden solicitar la misma ruta de funciones de servicio.

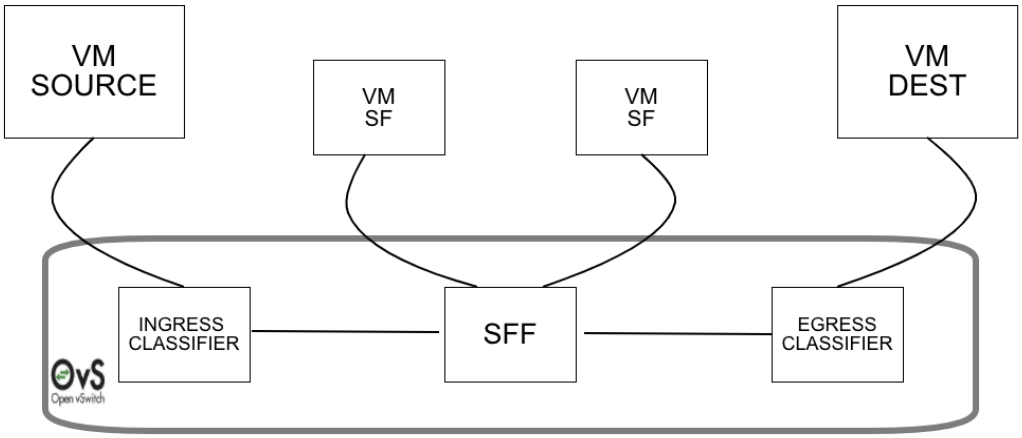


Figura 2.23: Flujo de paquetes en SFC

Como se puede apreciar en la figura 2.23, la arquitectura de SFC se compone de un clasificador de ingreso y otro de salida, estos clasificadores se encargan de clasificar los paquetes según las políticas de SFC en las distintas cadenas existentes, por ejemplo, se puede clasificar un paquete simplemente por el puerto de destino y las IPs, o se pueden clasificar por una tupla de 5 campos.

Una vez clasificados en una de las cadenas, los paquetes son redirigidos al SFF, que se encarga de enrutar los paquetes a la SF que corresponda según la encapsulación SFC, esta encapsulación va en el paquete, y es la que determina que SFP sigue el paquete. Una vez el paquete haya pasado por la SF, se devuelve al SFF, y este de nuevo enrutará el paquete a una nueva SF si corresponde, o dará por terminada la cadena, eliminando la encapsulación SFC y entregando el paquete al clasificador de salida, que devolverá el paquete a la red.

2.5.3. Proxy SFC

En ocasiones, las máquinas virtuales no son capaces de interpretar la encapsulación SFC (SFC unaware Function), para solventar esto, se inserta un SFC proxy entre una o más SFs y el SFF que les dirige el tráfico.

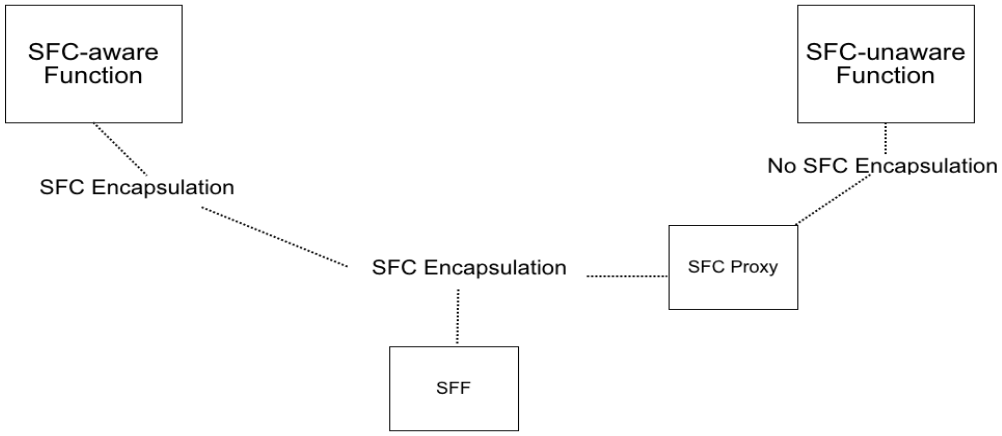


Figura 2.24: Proxy SFC

En la figura 2.24 se representa como un SFC Proxy se encarga de eliminar la encapsulación SFC de los paquetes recibidos por el SFF, para así poder entregar los paquetes a la unaware SF, de nuevo esta SF devuelve el paquete al Proxy que vuelve a añadir la encapsulación SFC y lo devuelve al SFF.

2.6. Marco regulador

2.6.1. Ley de protección de datos

El uso de servicios de Cloud Computing involucra un gran número de ventajas, pero a su vez, por sus características, unos riesgos específicos que involucran el tratamiento de datos.

Podemos agrupar estos riesgos en dos categorías principales:

- **Falta de transparencia**

  El prestador del servicio es el que conoce todos los detalles sobre sus servicios. Por lo tanto, el cliente se encuentra en la situación de saber qué, quién, cómo y dónde se están tratando los datos que el mismo está proporcionando.

- **Falta de control**

  Como consecuencia de el primer problema, surge la falta de control sobre los datos que proporciona el cliente, éste no puede saber la ubicación exacta de sus datos, se encuentra con dificultades a la hora de disponer de estos datos o de poder obtenerlos en un formato válido.

La normativa aplicable para el modelo de Cloud Computing viene dada por la Ley Orgánica 15/1999, de 13 de Diciembre, de Protección de Datos de Carácter Personal (LOPD) [22] que se encarga de la normativa aplicable al cliente y al prestador de servicio, en materia de protección de datos.

Algunos de los artículos mas destacables pueden ser:

- **Articulo 4.** *Calidad de los datos*

  1. Los datos de carácter personal sólo se podrán recoger para su tratamiento, así como someterlos a dicho tratamiento, cuando sean adecuados, pertinentes y no excesivos en relación con el ámbito y las finalidades determinadas, explícitas y legítimas para las que se hayan obtenido.
  2. Los datos de carácter personal serán cancelados cuando hayan dejado de ser necesarios o pertinentes para la finalidad para la cual hubieran sido recabados o registrados. No serán conservados en forma que permita la identificación del interesado durante un período superior al necesario para los fines en base a los cuales hubieran sido recabados o registrados.

- **Articulo 6.** *Consentimiento del afectado*

  1. El tratamiento de los datos de carácter personal requerirá el consentimiento inequívoco del afectado, salvo que la ley disponga otra cosa.
  4. En los casos en los que no sea necesario el consentimiento del afectado para el tratamiento de los datos de carácter personal, y siempre que una ley no disponga lo contrario, éste podrá oponerse a su tratamiento cuando existan motivos fundados y legítimos relativos a una concreta situación personal. En tal supuesto, el responsable del fichero excluirá del tratamiento los datos relativos al afectado.

- **Articulo 9.** *Seguridad de los datos*

  1. El responsable del fichero, y, en su caso, el encargado del tratamiento deberán adoptar las medidas de índole técnica y organizativas necesarias que garanticen la seguridad de los datos de carácter personal y eviten su alteración, pérdida, tratamiento o acceso no autorizado, habida cuenta del estado de la tecnología, la naturaleza de los datos almacenados y los riesgos a que están expuestos, ya provengan de la acción humana o del medio físico o natural.
  2. No se registrarán datos de carácter personal en ficheros que no reúnan las condiciones que se determinen por vía reglamentaria con respecto a su integridad y seguridad y a las de los centros de tratamiento, locales, equipos, sistemas y programas.
  3. Reglamentariamente se establecerán los requisitos y condiciones que deban reunir los ficheros y las personas que intervengan en el tratamiento de los datos a que se refiere el artículo 7 de esta Ley.

En este proyecto se evalúa a nivel técnico la infraestructura de una nube privada y la arquitectura de Service Function Chaining, pero los clientes, que son los que generan el tráfico, pueden enviar cualquier tipo de dato personal o información a través de la red.

2.6.2. RFC Service Function Chaining

También, es necesario respetar la arquitectura y el funcionamiento de SFC que se establece en la RFC del IETF, donde las principales consideraciones son las siguientes [18]

- Topológicamente independiente: No son necesarios cambios en la capa de reenvio para desplegar e invocar SFs o SFCs.
- Separación de planos: La realización de SFPs está separada del manejo de operaciones con paquetes.
- Clasificación: El tráfico debe cumplir las reglas de clasificación que dicta un SFP específico.
- Metadatos compartidos: Los metadatos pueden ser compartidos entre las SFs y entre redes externas y SFs (e.g. orquestación).
- Independencia de la definición de servicio: La arquitectua de SFC no depende de los detalles de cada SF.
- Independencia de la Service Function Chain: La creación, modificación o eliminación de una SFC no tiene impacto en otras SFCs.

Capítulo 3

Desarrollo de la solución técnica

En este capítulo se van a describir los pasos para llegar a la solución técnica, los requisitos que se han de cumplir, las razones por las que se deciden usar un entorno cloud para el desarrollo del proyecto, y otras posibles soluciones que se podrían aplicar.

3.1. Requisitos y entorno de desarrollo

En esta sección se describirán los requisitos para poder evaluar entornos de red sencillos, en los que se puedan apreciar como los paquetes generados desde una fuente, atraviesan los diferentes servicios de red de acuerdo a las reglas de SFC, y no de una manera tradicional. Estos entornos de red deben ser aproximaciones a aplicaciones que se podrían dar en entornos empresariales. El alcance del proyecto no permite simular sistemas que hagan uso de DPI, pero si se podrán simular sistemas que distribuyan los paquetes a servidores en función del tipo de tráfico, o firewalls sencillos, para ello será necesario el uso de distribuciones linux.

También se describirá cual es el entorno sobre el que se sostendrá el proyecto, y cuales son las razones tanto económicas como arquitectónicas por las cuales se decide entre un entorno u otro. Es importante destacar que este proyecto es un proyecto para evaluar el correcto funcionamiento de SFC sobre OpenStack. En un entorno empresarial la decisión puede ser totalmente diferente a la aquí propuesta por temas de escalabilidad, presupuesto o necesidades.

3.1.1. Requisitos de Hardware

Los requisitos de hardware que debe cumplir la plataforma sobre la que se instala el proyecto son los siguientes:

- **10 GB o más de memoria RAM**

  La instalación de OpenStack en un solo nodo requiere de unos 6 GB de memoria RAM, después se necesita memoria para asignar a las máquinas virtuales.

- **Ubuntu 14.04 o superior**

  A medida que se actualizan las dependencias de OpenStack, los requisitos de SO se actualizan.

- **20 GB disco duro**

  Con 20 GB se podrían levantar suficientes máquinas virtuales para probar los escenarios.

- **2 o más núcleos para funcionamiento óptimo**

  Con un solo núcleo la instalación funcionaría, pero retrasaría la velocidad de trabajo.

- **Conexión a internet de alta velocidad**

  Se necesita acceder a la máquina a través de internet y montar un servidor VNC.

- **Rango de IPs disponible**

  Las máquinas virtuales deben llevar asociadas unas IPs flotantes que deben estar libres en el rango de la subred.

3.1.2. Requisitos de OpenStack

Los requisitos que debe cumplir OpenStack son los siguientes:

- **Instanciar máquinas virtuales con CirrOS ó Ubuntu**

  Se necesitaran imagenes de lo sistemas operativos CirrOS y Ubuntu

- **Nova**
- **Glance**
- **Swift**
- **Neutron**
- **Cinder**
- **Keystone**
- **Horizon**
- **Service Function Chaining Plugin**

3.1.3. Requisitos de desarrollo

Los requisitos necesarios para poder desarrollar este proyecto:

- **Máquina con los requisitos de hardware descritos**

  La máquina para poder desarrollar el proyecto es indispensable y debe cumplir todos los requisitos.

- **Poder trabajar en remoto desde cualquier lugar**

  Se debe poder trabajar en remoto a través del protocolo SSH.

- **Total disponibilidad horaria del entorno de trabajo**

  El horario para poder desarrollar el proyecto puede variar de un dia a otro, por lo que se necesita poder acceder a este en cualquier momento.

- **Entorno Cloud donde poder desarrollar el proyecto**

  Ya que el proyecto se va a desarrollar en un entorno Cloud, se necesita alquilar una máquina a un proveedor Cloud.

3.1.4. Entorno de desarrollo

De acuerdo con los requisitos descritos anteriormente, para desarrollar este proyecto se necesita una máquina con unos requisitos un tanto elevados, sobre todo en memoria RAM, las opciones son dos, una máquina física que habría que adquirir, o un entorno cloud donde solo cobraría por el tiempo que tengamos la máquina en uso.

Actualmente, un ordenador con estas características costaría en torno a los 800 Euros, visto que además necesitamos poder acceder a esta máquina desde cualquier lugar, el ordenador tendría que estar constantemente encendido, con su correspondiente gasto de electricidad. En este proyecto se apuesta por la puesta en escena de entornos cloud y la virtualización en detrimento de las tecnologías convencionales, es por ello que la mejor opción para desarrollar un entorno de pruebas, es la de alquilar una máquina con las características que necesitemos a un proveedor Cloud.

En este proyecto se ha decidido alquilar una máquina a la empresa Google, que ofrece sus servicios en la nube a través de la plataforma Google Cloud. Esta solución cumple a la perfección con todos nuestros requisitos y por lo tanto será el entorno de desarrollo utilizado.

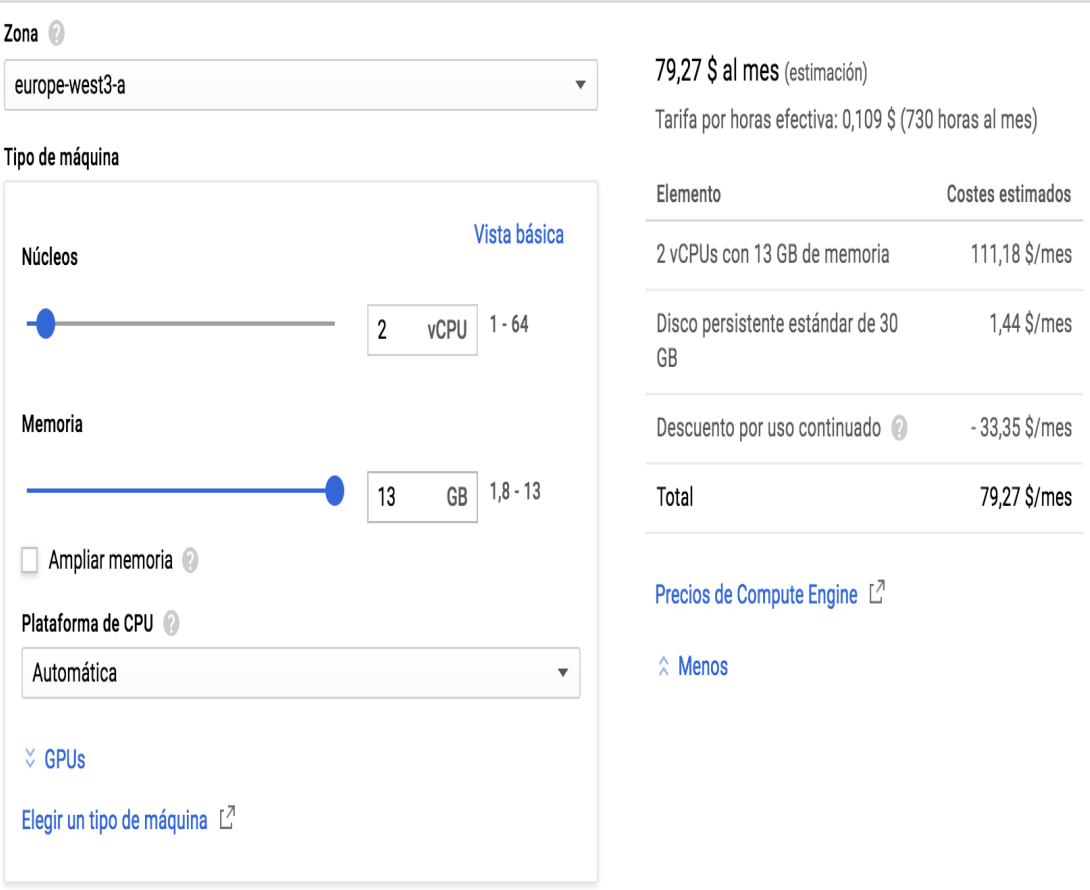


Figura 3.1: Precio de servicio Cloud

Se puede observar en la figura 3.1 como el precio es bastante reducido para una máquina de unas prestaciones relativamente altas. Los 79 dolares son por un uso continuado de 24 horas al día, lo cual, en este proyecto no va a ser así, se estima que se necesitaran unas 100 horas al mes y con eso será suficiente.

En el siguiente gráfico se puede apreciar la evolución del precio del alquiler de un servicio Cloud, frente al coste de una máquina de las mismas características, a lo largo de los 12 meses que podría durar el proyecto completo en todas sus fases durante 100 horas por mes.

Como se puede apreciar en la tabla 3.1, el ahorro es considerable, y harían falta mas de cinco años para empezar a rentabilizar la máquina física. Cinco años en los cuales la tecnología puede avanzar a pasos agigantados y los requisitos quedarse cortos, por lo cual habría que renovar la máquina, una desventaja enorme teniendo en cuenta que en el servicio Cloud, podemos aumentar las capacidades en cualquier momento y sin un incremento de precio excesivo.

	MES											
	1	2	3	4	5	6	7	8	9	10	11	12
Pc 12 gb ram	800	800	800	800	800	800	800	800	800	800	800	800
Google Cloud	10,9	21,8	32,7	43,6	54,5	65,4	76,3	87,2	98,1	109	119,9	130,8

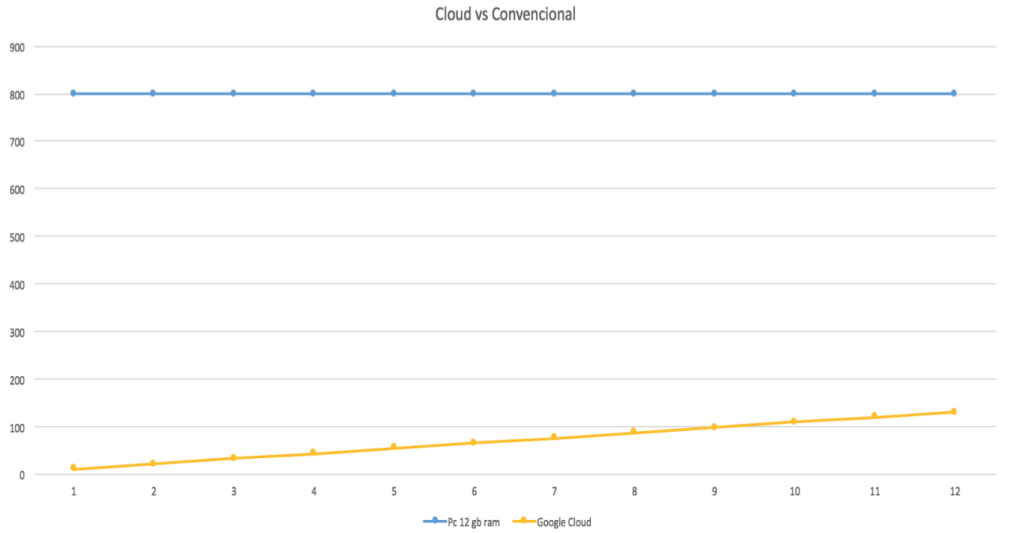


Tabla 3.1: Comparación del precio de comprar un PC con el alquiler de un servicio Cloud a lo largo de doce meses

3.2. Diseño

En esta sección se procede a describir el diseño y los pasos a seguir para desplegar la solución técnica, sobre la que posteriormente se procederá a evaluar las pruebas de red.

El procedimiento se explica paso a paso, y se describe el significado de cada línea de los ficheros de configuración, además de la arquitectura interna en algunos casos.

Mediante esta sección se puede llegar a una solución exactamente igual a la de este proyecto para quien sea que quiera probar el entorno, y así poder realizar sus propias pruebas o reestructurar el proyecto para implantar nuevas tecnologías que están surgiendo en el paradigma Cloud.

3.2.1. DevStack: primeros pasos

Figura 3.2: DevStack logo

DevStack [27] es una serie de scripts personalizables, usados para proveer de un entorno OpenStack rápidamente, basado en las ultimas versiones de todos los módulos que existen en la rama master del repositorio Git. Se utiliza interactivamente como un entorno de desarrollo y como base para gran parte de las pruebas funcionales del proyecto OpenStack.

DevStack ahorra gran parte del tiempo en este proyecto, ya que una instalación multinodo de un entorno OpenStack de producción es muy costosa y compleja, además de requerir más recursos económicos y técnicos. Por lo tanto, DevStack se ajusta a la perfección para desplegar un entorno de pruebas y en futuros proyectos poder pasar estas pruebas a producción.

En las siguientes subsecciones, se explicarán los pasos para descargar DevStack con todos sus scripts.

3.2.1.1. Instalar Linux

Empezar con una instalación limpia y mínima de un sistema Linux. DevStack soporta desde Ubuntu 14 en adelante, Fedora 24/25, CentOS/RHEL 7, también como Debien y OpenSUSE.

Para este proyecto se instalará Ubuntu 16.04, la distribución más testeada.

3.2.1.2. Añadir usuario Stack

DevStack debe ser ejecutado por un usuario sin ser root pero con los permisos sudo habilitados.

Se puede crear un usuario stack para correr DevStack con el siguiente comando

```
sudo useradd -s /bin/bash -d /opt/stack -m stack
```

Ya que el usuario va a hacer bastantes cambios en el sistema, debe tener privilegios de superusuario

```
echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
sudo su - stack
```

3.2.1.3. Descargar DevStack

El repositorio DevStack contiene una serie de scripts para instalar OpenStack, y unas plantillas de configuración

```
git clone https://git.openstack.org/openstack-dev/devstack
cd devstack
```

3.2.2. Local.conf

Para la configuración de DevStack se utiliza un fichero llamado local.conf [26]. Es un fichero INI modificado que contiene unas cabeceras que llevan asociadas ciertos parámetros de los ficheros de configuración que deben ser cambiados.

La cabecera es similar a una sección INI normal, pero con la particularidad de los dobles corchetes ([[....]]), y dos campos internos separados por una barra (—),

```
[[<phase>|<config-file-name>]]
```

Phase es una de las fases definidas en el script stack.sh y config-file-name es el fichero de configuración a modificar.

Las fases definidas son las siguientes:

- local - Extrae el localrc del local.conf antes de que stackrc sea llamado
- post-config - Se ejecuta después de que los servicios de la capa 2 estén configurados y antes de que sean iniciados
- extra - Se ejecuta después de que los servicios son iniciados y antes de que ningún archivo en extra.d sea ejecutado
- post-extra - Se ejecuta después de que los archivos en extra.d sean ejecutados

El archivo es procesado en orden, es decir, los parámetros pueden ser duplicados, pero siempre tendrá efecto el último que aparezca.

3.2.2.1. openrc

openrc configura las credenciales de autenticación adecuadas para el uso de la linea de comandos de OpenStack, openrc ejecuta stackrc al inicio para así poder acceder a los campos HOST _IP y SERVICE_HOST.

Los siguientes valores son los valores por defecto

```
OS_PROJECT_NAME=demo
OS_USERNAME=demo
OS_PASSWORD=secret
HOST_IP=127.0.0.1
SERVICE_HOST=$HOST_IP
OS_AUTH_URL=http://$SERVICE_HOST:5000/v2.0
# export KEYSTONECLIENT_DEBUG=1
# export NOVACLIENT_DEBUG=1
```

3.2.2.2. Directorio de instalación

El directorio de instalación de DevStack va definido en la variable DEST. Por defecto es /opt/stack

```
DEST=/opt/stack
```

3.2.2.3. Logging

Por defecto, la salida de stack.sh solo es escrita en la consola cuando está corriendo. Puede ser enviada a un archivo externo mediante la variable LOGFILE

```
LOGFILE=$DEST/logs/stack.sh.log
```

Los logs antiguos pueden ser eliminados mediante la variable LOGDAYS, que se deshace de los logs mas antiguos que el número de días que indique

```
LOGDAYS=1
```

3.2.2.4. Instalación limpia cada vez

Por defecto, stack.sh solo clona los proyectos del repositorio si no existen en la carpeta de instalación. Si la variable RECLONE esta activada, stack.sh comprobará si existen versiones más actualizadas de cada proyecto.

```
RECLONE=yes
```

3.2.2.5. Actualizar paquetes instalados por pip

Por defecto, stack.sh solo instala los paquetes Python si estos no están instalados, o la versión actual no coincide con lo especificado en los requisitos. Si PIP_UPGRADE está establecida en True, entonces todos los paquetes Python necesarios serán actualizados a las versiones mas recientes.

```
PIP_UPGRADE=True
```

3.2.2.6. Imágenes externas

Las imágenes proporcionadas por url a través de la variable IMAGE_URLS serán descargadas y subidas a glance.

Las imágenes por defectos son predefinidas por cada tipo de hipervisor. Estableciendo DOWNLOAD_DEFAULT_IMAGES=False prevendrá a DevStack de descargar estas imágenes. En este caso, es recomendable rellenar la variable IMAGE_URLS con otras imágenes que se necesiten.

```
DOWNLOAD_DEFAULT_IMAGES=False
IMAGE_URLS="http://foo.bar.com/image.qcow,"
IMAGE_URLS+="http://foo.bar.com/image2.qcow"
```

3.2.2.7. Configuración mínima

Aquí se propone un ejemplo de un local.conf con la configuración mínima para poder arrancar.

- Sin logs
- Contraseñas introducidas
- La IP del host introducida
- Rangos de red desplazados fuera de la red local

```
[[local|localrc]]
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
#IPV4_ADDRS_SAFE_TO_USE=172.31.1.0/24
#FLOATING_RANGE=192.168.20.0/25
#HOST_IP=10.3.4.5
```

Si las variables de contraseña no están definidas, stack.sh se detendrá para pedirlas por teclado.

Los rangos de red no se deben solapar con las IPs que ya están en uso.

HOST_IP es normalmente detectada por stack.sh, pero a menudo no se puede determinar en futuros inicios, ya que la IP puede haber sido desplazada desde una interfaz Ethernet a un bridge en el host. Es recomendable especificarla aquí.

3.2.3. Crear local.conf

En esta sección se creará un local.conf que se ajuste a las necesidades de este proyecto y sus módulos necesarios.

```
# Sample ‘‘local.conf‘‘ for SFC scenarios in ‘‘stack.sh‘‘

# NOTE: Copy this file to the root DevStack directory for it to work
    properly.

# ‘‘local.conf‘‘ is a user-maintained settings file that is sourced from
     ‘‘stackrc‘‘.
# This gives it the ability to override any variables set in ‘‘stackrc
    ‘‘.
# Also, most of the settings in ‘‘stack.sh‘‘ are written to only be set
    if no
# value has already been set; this lets ‘‘local.conf‘‘ effectively
    override the
# default values.

# This is a collection of some of the settings I have found to be useful
# in our DevStack development environments. Additional settings are
    described
# in http://docs.openstack.org/developer/devstack/configuration.html#
    local-conf

# The ‘‘localrc‘‘ section replaces the old ‘‘localrc‘‘ configuration
    file.
# Note that if ‘‘localrc‘‘ is present it will be used in favor of this
    section.
[[local|localrc]]

RECLONE=no
HOST_IP=10.132.0.2
FLOATING_RANGE=10.4.0.0/27
FIXED_RANGE=10.0.2.0/24
FIXED_NETWORK_SIZE=256
FLAT_INTERFACE=eth0
ADMIN_PASSWORD=javi
DATABASE_PASSWORD=javi
RABBIT_PASSWORD=javi
SERVICE_PASSWORD=javi

#FIX THE PROBLEM SETKEYCODES 00 FOR CIRROS IMAGE
NOVNC_BRANCH=v0.6.0

disable_service n-net
enable_service q-dhcp q-svc q-agt q-l3 q-meta q-qos neutron

Q_PLUGIN=ml2
```

```
37 Q_AGENT=openvswitch
38 ENABLE_TENANT_TUNNELS=True
39 Q_ML2_TENANT_NETWORK_TYPE=vxlan
40
41 IMAGE_URLS="https://bugs.launchpad.net/fuel/+bug/1260268/+attachment
       /4637521/+files/disk-1.img,"
42 IMAGE_URLS+="http://cloud-images.ubuntu.com/xenial/current/xenial-server
       -cloudimg-amd64-disk1.img,"
43 IMAGE_URLS+="https://download.fedoraproject.org/pub/fedora/linux/
       releases/25/CloudImages/x86_64/images/Fedora-Cloud-Base-25-1.3.x86_64
       .qcow2"
44 #SFC
45 enable_plugin networking-sfc https://git.openstack.org/openstack/
       networking-sfc
46 SFC_UPDATE_OVS=FALSE
47
48 LOGFILE=$DEST/logs/stack.sh.log
49 LOGDAYS=2
```

- Línea 19: Dado que DevStack tiene tendencia a actualizarse cada poco tiempo, es mejor dejar siempre la misma versión para evitar problemas de compatibilidad.
- Línea 20: Dado que se tiene asignada una IP fija en Google Cloud, se especifica aquí.
- Línea 21: Se asigna un rango de IPs flotantes que no esté siendo utilizado.
- Línea 22: Rango de IPs fijas que asigna Nova mediante DHCP.
- Línea 31: Debido a un error con los teclados europeos y la consola de CirrOS hay que especificar otra versión de novnc.
- Línea 33: Se desactiva el servicio de networking que ofrece Nova.
- Línea 34: Se activan los servicios de Neutron.
- Línea 36 37 38: Se especifican las opciones de tuneles para el funcionamiento de SFC.
- Línea 40: Se descarga una imagen de CirrOS con tcpdump habilitado, una imagen de xenial ubuntu y una Fedora.
- Línea 44: Se descarga el servicio de SFC.
- Línea 45: Se especifica que no se reconstruyan las tablas de OVS.

Con este local.conf , colocándolo en la raiz de la carpeta de devstack creada anteriormente, ya se puede arrancar la instalación de DevStack.

3.2.4. Diferencias entre Floating y Fixed IPs

En el local.conf es necesario especificar los rangos de las IPs flotantes y las IPs fijas.

Mientras que las IPs fijas son agregadas a las instancias por defecto, no garantizan que la instancia sea inmediatamente accesible desde el internet .exterior"(o desde el resto del data center). Tomando como ejemplo el siguiente escenario:

Corres un pequeño sitio web LAMP con un servidor www, una base de datos, y un firewall que se encarga de la traducción de direcciones de red (NAT) y el filtro de tráfico. Tipicamente uno desea aplicar lo siguiente:

- Todos los servidores se comunican internamente en un rango de red (no routable) (e.g. 192.168.0.0/24).
- Hay una IP pública enrutable en la cual el servidor www es visible.

Se hace lo siguiente:

- Configurar el firewall con la IP pública.
- Crear una regla en el NAT para reenviar el tráfico de la IP pública hacia la privada en el servidor www.

Las IPs fijas en OpenStack funcionan de la misma manera que el rango 192.168.0.0/24 en el ejemplo anterior. Solo garantizan conexión entre instancias dentro de OpenStack.

Pero OpenStack también introduce otro tipo de direcciones IP, llamadas IPs flotantes. Las IPs flotantes son simplemente IPs públicas que típicamente se adquieren a un ISP. Los usuarios pueden asignarlas a sus instancias, haciéndolas alcanzables desde el mundo exterior.

Las IPs flotantes no son asignadas a las instancias por defecto. Los usuarios del Cloud necesitan explícitamente cogerlas del pool configurado por el administrador de OpenStack, y después vincularlas a sus instancias. Una vez el usuario ha cogido una IP flotante del pool, se convierte en dueño exclusivo de esa IP, si la instancia cae por algún motivo, el usuario no pierde la IP, permanece en sus propios recursos para volver a ser asignada a otra instancia.

Por otro lado, las IPs fijas son asignadas dinámicamente por nova-network (en este proyecto se desactiva y se asigna esta tarea a q-dhcp) cuando las instancias se crean. No hay ninguna manera de decirle a OpenStack que asigne una IP fija específica a una instancia, por lo tanto, cuando una máquina virtual termina accidentalmente y se restaura desde una snapshot, la nueva instancia arrancará con otra IP fija.

Los administradores de OpenStack pueden configurar múltiples pools de IPs flotantes, sin embargo, al contrario que los pools de IPs fijas, las IPs flotantes

no pueden ser mapeadas para proyectos específicos (tenants). Cada usuario puede coger una IP flotante de cualquier pool de IPs flotantes que quiera.

La principal motivación detras de los diferentes pools de IPs flotantes, es la de que cada uno de ellos puede ser proporcionado por diferentes ISPs, de esta manera se puede asegurar que se mantiene la conexión incluso aunque uno de los ISPs se caiga durante unos momentos.

Simplemente para resumir, estas son las claves de las IPs flotantes:

- Las IPs flotantes no son asignadas a las instancias por defecto.
- Si una instancia cae, el usuario puede reutilizar la misma IP flotante reasignandola a otra instancia.
- Los usuarios pueden coger IPs flotantes de diferentes pools definidos por el administrador del Cloud para asegurar conectividad desde diferentes ISPs.

Dado que en este proyecto no se contempla la posibildad de adquirir IPs a ningún ISP, el rango de IPs flotantes solo será accesible desde el entorno de desarrollo, la figura 3.3[1], muestra un ejemplo configuración en un data center donde las IPs flotantes solo son accesibles por otros servicios dentro de la misma red.

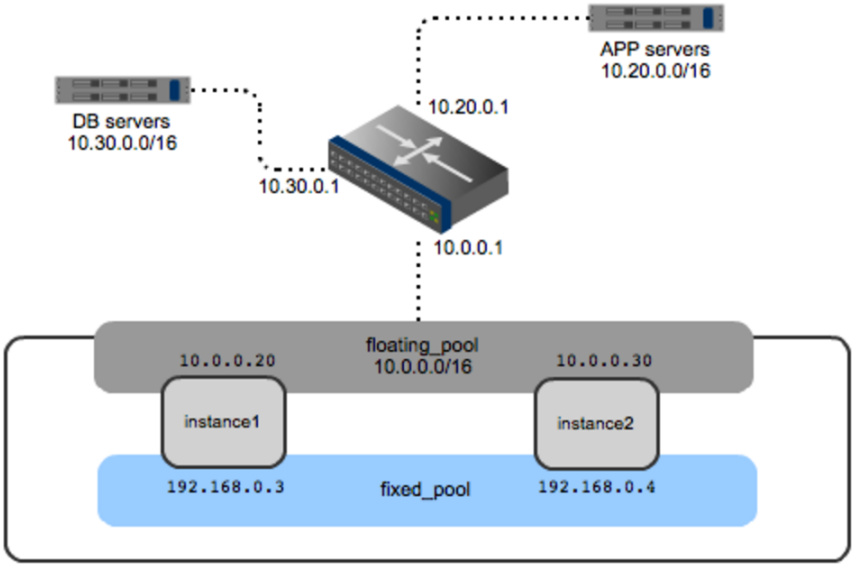


Figura 3.3: Ejemplo de configuración de red en un Data Center

[1]https://www.mirantis.com/blog/configuring-floating-ip-addresses-networking-openstack-public-private-clouds/

3.2.5. Arrancando DevStack

Una vez situado el local.conf, y habiendo descargado la última versión estable de DevStack, hay que situarse en la carpeta devstack creada y empezar la instalación.

La carpeta devstack debe tener esta estructura

Figura 3.4: Carpeta devstack

Se ejecuta el siguiente comando

```
./stack.sh
```

La primera instalación puede durar alrededor de 80 minutos, dependiendo de las características de la máquina, ya que tiene que descargar todos los componentes de OpenStack y crear y cofigurar las base de datos.

Si todo ha ido según lo esperado, esta debería ser la salida del comando.

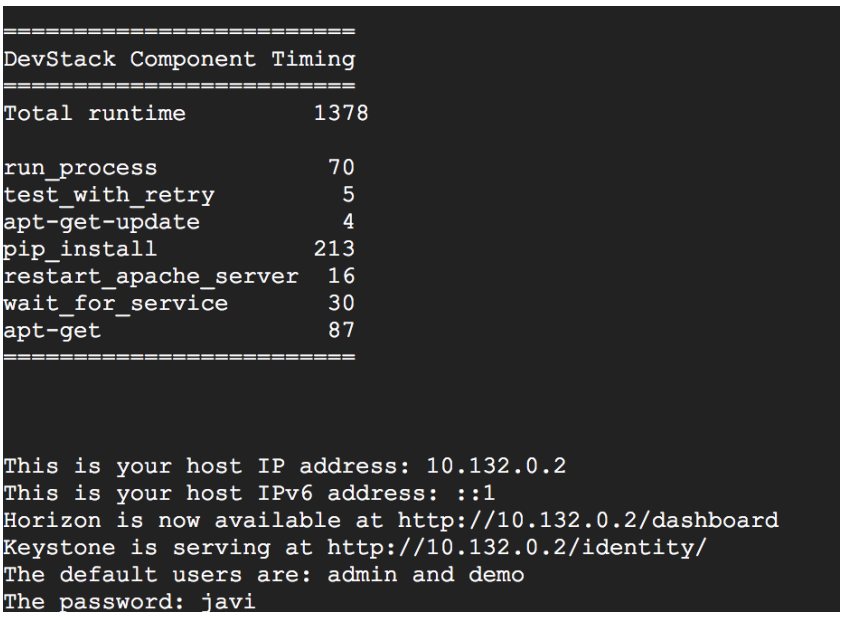


Figura 3.5: Tiempo de instalación de DevStack

Una vez arrancado DevStack, se puede parar el entorno Cloud con el siguiente comando

```
./unstack.sh
```

O bien desinstalarlo con este otro

```
./clean.sh
```

Por el momento no es necesario hacer ninguna de las dos cosas.

Para acceder a horizon es necesario abrir el navegador, y acceder a la dirección IP local localhost, este será el panel de login.

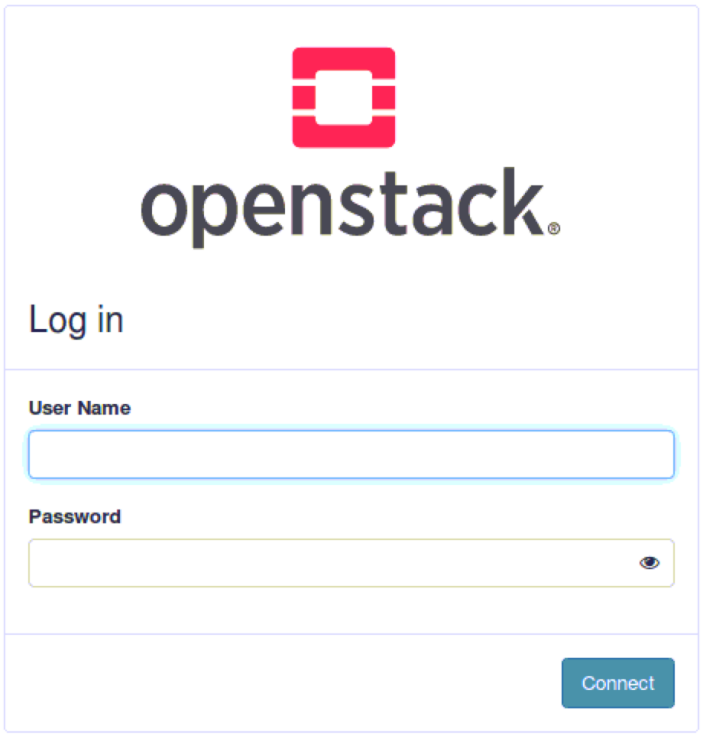


Figura 3.6: Panel de autenticación de OpenStack

DevStack instala por defecto defecto el proyecto demo, por lo que se puede acceder mediante dos maneras:

- Usuario normal del proyecto demo con usuario demo y contraseña javi.
- Administrador con acceso a todos los proyectos, usuario admin y contraseña javi.

3.2.6. Screens

Dado que la instalación se hace en único nodo, debe existir alguna manera de poder acceder a los logs y comandos que está generando cada módulo de OpenStack, DevStack soluciona esto gracias al uso de las screens de linux. Por defecto se crea una screen llamada 'stack' con x ventanas, cada ventana representa el proceso/logs de los componentes de OpenStack.

Screen es un gestor de screens que multiplexa un terminal entre varios procesos, típicamente shells interactivas.

- Mostrar las screens disponibles

```
$ screen -ls
There is a screen on:
        29208.stack    (08/16/2017 12:48:10 PM)     (Detached)
1 Socket in /var/run/screen/S-javi.
```

- Anclarse con la screen de devstack

```
$ screen -x stack
```

Aparecerá una pantalla similar a esta

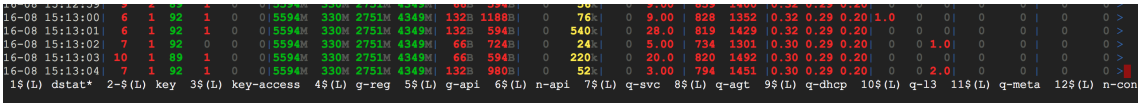


Figura 3.7: Screens Devstack

Algunos atajos importantes son:

- CTRL + A + UP/DOWN FLECHA Para cambiar entre las screens que aparecen en la parte inferior
- CTRL + A + n Para pasar a la siguiente screen
- CTRL + A + Para pasar a la screen previa
- CTRL + D Para salir de Screen

Los servicios que aparecen son los siguientes:

- key, key-access son para los componentes de Keystone, muestran los logs de éste.
- g-reg, g-api son componentes de Glance.
- n-api, n-cond, n-sch, n-novnc, n-cauth, n-cpu son componentes de Nova. n-api ejecuta los procesos de nova-api, n-cond ejecuta los procesos de nova-conductor, n-sch se ejecuta junto con nova-scheduler, n-novnc se ejecuta junto con nova-vnc, n-cauth se ejecuta junto con el módulo auth de nova, n-cpu se ejecuta junto con nova compute.
- q-svc,q-agt,q-dhcp,q-l3,q-meta son componentes de Neutron.
- c-api,c-vol,c-sch con componentes de Cinder.
- horizon es Horizon

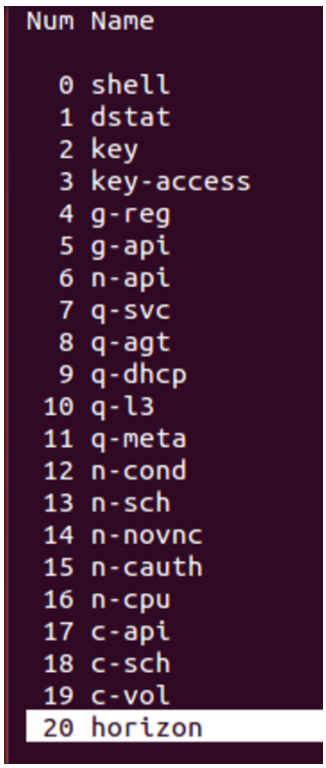


Figura 3.8: Servicios de DevStack

3.2.7. Uso de OpenStack

En el menú de instancias de OpenStack, se pueden levantar máquinas virtuales bien a partir de imágenes, snapshots de imágenes, volúmenes o snapshots de volúmenes (figura 3.9).

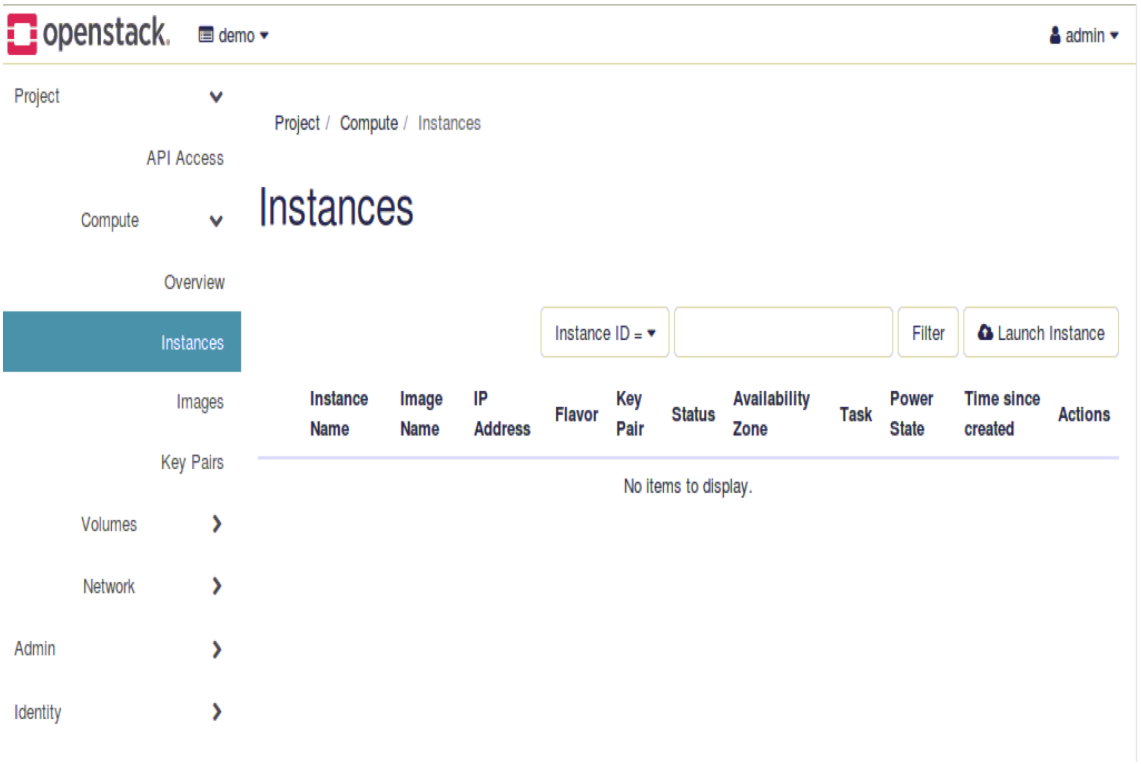


Figura 3.9: Menú instancias de OpenStack

En el caso de este proyecto, las instancias se levantarán a partir de imágenes de CirrOS y Ubuntu, para poder simular el comportamiento de un firewall.

También es posible iniciar instancias a partir de un volumen, y en este hacer un mount para poder iniciar un sistema operativo, pero esta opción no se tomará.

A su vez es posible la creación de routers, asociándolos a subredes que es posible crear a partir de un pool de direcciones de red que se haya proporcionado.

El usuario admin, tiene privilegios para poder editar los usuarios, crearlos, asignar roles, editar cualquier proyecto, a su vez también es posible poder editar las imágenes, los "flavoursz los parámetros de nova asignados por defecto.

3.2.7.1. CLI OpenStack

Para el uso de OpenStack, muchas veces es más ventajoso el uso de la línea de comandos [25] que el uso de Horizon, en especial a la hora de crear scripts que permitan hacer despliegues de entornos de prueba, ya que en este proyecto no se usarán las plantillas del orquestador Heat ni ningún tipo de tacker.

Para explicar los comandos más típicos, se propone un ejemplo donde estos son los objetivos:

- Crear un proyecto llamado white.
- Crear un usuario llamado 'white' y asociarlo al proyecto white con rol Miembro.
- Asociar el usuario admin como rol administrador en el proyecto white
- Crear una red privada llamada WHITE-PRIVATE1 con una subred WHITE-SUBNET1 192.168.101.0/24
- Crear un grupo de seguridad 'white-sec-group' para permitir el tráfico TCP, ICMP en ambos lados.
- Crear una instancia whiteinstace1 con una imágen CirrOS, 64MB RAM, red WHITE-PRIVATE1, white-sec-group.

```
javi@sfc:~/devstack$ source openrc admin admin
WARNING: setting legacy OS_TENANT_NAME to support cli tools.
javi@sfc:~/devstack$ openstack project create --description "White
    Tenant" white
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | White Tenant                     |
| domain_id   | default                          |
| enabled     | True                             |
| id          | c3cf5f816d294708a69eef02acb98d0e |
| is_domain   | False                            |
| name        | white                            |
| parent_id   | default                          |
+-------------+----------------------------------+
javi@sfc:~/devstack$ openstack project list
javi@sfc:~/devstack$ openstack user create --password white123 white
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 670406df54ab495ea5df33194241ccea |
| name                | white                            |
| options             | {}                               |
```

```
| password_expires_at | None                             |
+---------------------+----------------------------------+
javi@sfc:~/devstack$ openstack role add --project white --user white
    Member
javi@sfc:~/devstack$ openstack role add --project white --user admin
    admin
javi@sfc:~/devstack$ source openrc white white
WARNING: setting legacy OS_TENANT_NAME to support cli tools.
export OS_PASSWORD=white123
javi@sfc:~/devstack$ openstack network create WHITE-PRIVATE1
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2017-08-23T10:33:09Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | c69ea07f-2803-4af7-a916-41cfa43fa527 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | None                                 |
| mtu                       | 1450                                 |
| name                      | WHITE-PRIVATE1                       |
| port_security_enabled     | True                                 |
| project_id                | c3cf5f816d294708a69eef02acb98d0e     |
| provider:network_type     | None                                 |
| provider:physical_network | None                                 |
| provider:segmentation_id  | None                                 |
| qos_policy_id             | None                                 |
| revision_number           | 3                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| updated_at                | 2017-08-23T10:33:09Z                 |
+---------------------------+--------------------------------------+
javi@sfc:~/devstack$ openstack subnet create --subnet-range
    192.168.101.0/24 --dhcp --gateway 192.168.101.1 --network WHITE-
    PRIVATE1 WHITE-SUBNET1
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 192.168.101.2-192.168.101.254        |
| cidr              | 192.168.101.0/24                     |
| created_at        | 2017-08-23T10:33:54Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
```

```
| enable_dhcp     | True                                 |
| gateway_ip      | 192.168.101.1                        |
| host_routes     |                                      |
| id              | 74380033-2ef8-4e0d-9cd6-7e0c7c21ec49 |
| ip_version      | 4                                    |
| ipv6_address_mode | None                               |
| ipv6_ra_mode    | None                                 |
| name            | WHITE-SUBNET1                        |
| network_id      | c69ea07f-2803-4af7-a916-41cfa43fa527 |
| project_id      | c3cf5f816d294708a69eef02acb98d0e     |
| revision_number | 2                                    |
| segment_id      | None                                 |
| service_types   |                                      |
| subnetpool_id   | None                                 |
| updated_at      | 2017-08-23T10:33:54Z                 |
+-----------------+--------------------------------------+
javi@sfc:~/devstack$ openstack security group create --description "
    white security group" white-sec-group
| rules |direction='egress', ethertype='IPv4', id='c5c7b17a-b9e1-4c3f-9
    b1d-bf467f03af5d', revision_number='1'
|         |direction='egress', ethertype='IPv6', id='1e482eb9-2fbd-44cd
    -9e33-856270fa185f', revision_number='1'

javi@sfc:~/devstack$ openstack security group rule delete c5c7b17a-b9e1
    -4c3f-9b1d-bf467f03af5d
javi@sfc:~/devstack$ openstack security group rule delete 1e482eb9-2fbd
    -44cd-9e33-856270fa185f
javi@sfc:~/devstack$ openstack security group rule create --protocol
    icmp --egress white-sec-group
javi@sfc:~/devstack$ openstack security group rule create --protocol tcp
     --ingress white-sec-group
javi@sfc:~/devstack$ openstack security group rule create --protocol tcp
     --egress white-sec-group
javi@sfc:~/devstack$ openstack security group rule create --protocol
    icmp --ingress white-sec-group
openstack server create --image cirros-0.3.5-x86_64-disk --flavor m1.
    nano --security-group white-sec-group --nic net-id=c69ea07f-2803-4af7
    -a916-41cfa43fa527 whiteinstance1
```

Una vez introducidos estos comandos, los objetivos del ejemplo quedan cumplidos.

A continuación se resumen los comandos más importantes utilizados en el ejemplo:

```
source openrc admin admin
```

Source openrc permite cambiar entre proyecto y usuario de una manera sencilla

```
openstack user create --password white123 white
```

Con user create permite la creación de usuarios

```
openstack server create --image cirros-0.3.5-x86_64-disk --flavor m1.
    nano --security-group white-sec-group --nic net-id=c69ea07f-2803-4af7
    -a916-41cfa43fa527 whiteinstance1
```

Uno de los comandos que más se utilizará será el de crear instancias

```
openstack network create WHITE-PRIVATE1
```

network create permite crear redes

```
openstack subnet create --subnet-range 192.168.101.0/24 --dhcp --gateway
     192.168.101.1 --network WHITE-PRIVATE1 WHITE-SUBNET1
```

subnet create permite crear subredes de redes creadas anteriormente

```
openstack security group create --description "white security group"
    white-sec-group
```

security group create permite crear grupos de seguridad que actúan a modo de firewall para los paquetes entrantes y salientes de las instancias.

Es importante el detalle de que todos los comandos salvo el de cambio de proyecto, van introducidos por la palabra openstack en las nuevas versiones, es posible encontrar comandos precedidos por el servicio que se vaya a utilizar (e.g. nova, neutron, keystone), pero estos actualmente están obsoletos y no deberían utilizarse salvo que no hayan sido actualizados.

3.2.7.2. Línea de comandos de SFC

Los comandos de Service Function Chaining funcionan sobre la línea de comandos de neutron, ya que las operaciones van asociadas a creación de puertos y direccionamiento de paquetes.

A continuación se describen los existentes hasta la fecha:

```
$ neutron port-create --name p1 net1
```

neutron port-create nos permite crear puertos que serán asociados a las instancias más tarde, por estos puertos entrarán y saldrán los paquetes que se comunicarán con el SFF.

```
$ openstack server create --nic port-id=P1_ID --nic port-id=P2_ID vm1
```

A la hora de crear una instancia, se le asignaran los puertos creados anteriormente como se aprecia en la figura 3.10, y solo los que sean necesarios.

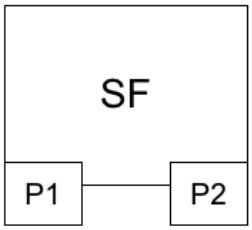


Figura 3.10: Instancia con dos puertos

```
$ neutron flow-classifier-create \
  --description "Description of the flow classifier" \
  --ethertype IPv4 \
  --source-ip-prefix 10.0.0.2/32 \
  --destination-ip-prefix 10.0.0.3/32 \
  --protocol tcp \
  --source-port 1000:1000 \
  --destination-port 80:80 FC1
```

Con neutron flow-classifier-create se crea el clasificador de paquetes, que describe los paquetes que seguirán la cadena, en función de las direcciones origen y destino, el protocolo y los puertos como se ve en la figura 3.13.

```
$ neutron port-pair-create \
  --description "Instance 1" \
  --ingress p1 \
  --egress p2 PP1
```

Con neutron port-pair-create se crean parejas de puertos entrada y salida como en la figura 3.11.

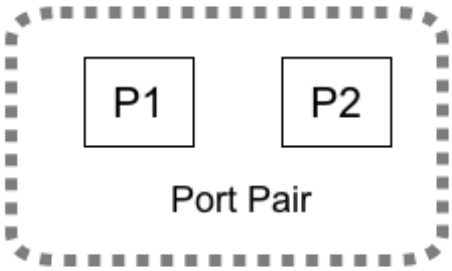


Figura 3.11: Port Pair

```
$ neutron port-pair-group-create \
  --port-pair PP1 --port-pair PP2 PPG1\
$ neutron port-pair-group-create \
  --port-pair PP3 PPG2
```

Con neutron port-pair-group-create es posible la creación de grupos de pares de puertos (figura 3.12) que más adelante se explicará su funcionamiento.

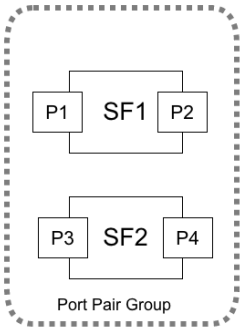


Figura 3.12: Port Pair Group

```
$ neutron port-chain-create \
  --port-pair-group PPG1 --port-pair-group PPG2 \
  --flow-classifier FC1 PC1
```

Con neutron port-chain-create se crea la cadena de servicio, que contiene los grupos de pares de puertos asociados a las instancias por los que pasarán los paquetes, estos grupos pueden contener uno o más pares de puertos (figura 3.13).

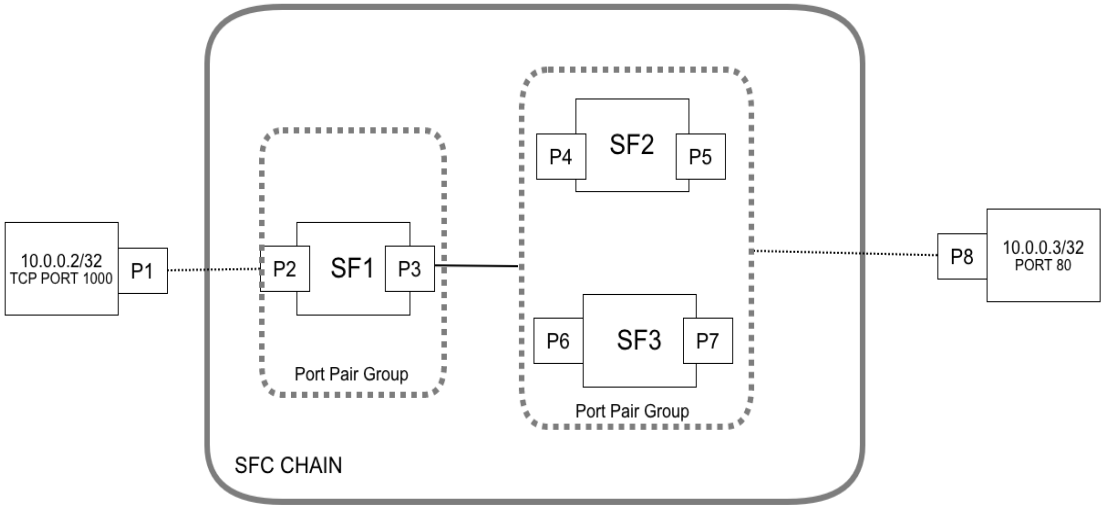


Figura 3.13: Cadena de SFC

```
$ neutron port-chain-update \
  --port-pair-group PPG1 --port-pair-group PPG2 --port-pair-group PPG3 \
  --flow-classifier FC1 PC1
```

Es posible añadir un grupo de pares de puertos a la cadena una vez creada mediante neutron port-chain-update.

```
$ neutron port-chain-update \
  --port-pair-group PPG1 --port-pair-group PPG2 \
  --flow-classifier FC1 --flow-classifier FC2 PC1
```

A su vez también es posible añadir un nuevo clasificador a la cadena.

Esta es la lista completa de comandos en SFC

```
flow-classifier-create
flow-classifier-delete
flow-classifier-list
flow-classifier-show
flow-classifier-update

port-pair-create
port-pair-delete
port-pair-list
port-pair-show
port-pair-update

port-pair-group-create
port-pair-group-delete
port-pair-group-list
port-pair-group-show
port-pair-group-update

port-chain-create
port-chain-delete
port-chain-list
port-chain-show
port-chain-update
```

3.2.7.3. Port Pair Group en SFC

Para poder crear una cadena de servicio en SFC, es necesario crear esa cadena indicando los grupos de parejas de puertos por los que van a pasar los paquetes.

Estos grupos pueden estar formados por una o mas parejas de puertos, pudiendo así hacer distribución o balanceo de carga sobre estas parejas.

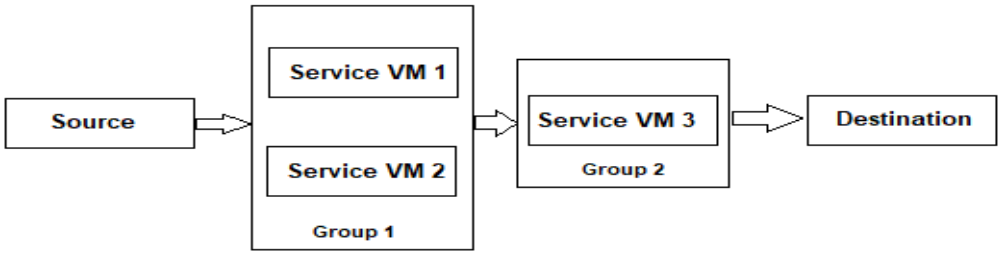


Figura 3.14: Port Pair Group

Como se puede observar en la figura 3.10, el grupo 1 está compuesto por dos funciones, por lo que los paquetes pasaran o bien por una o bien por otra función, pero nunca por las dos a la vez, consiguiendo el efecto de balanceo de carga.

Este balanceo de carga [29] es un tanto engañoso y es una de las propuestas de mejora para futuras versiones de OpenStack. La razón de todo esto es que para la implementación OVS, el balanceo de carga de tráfico hacia diferentes port-pairs (Service Functions) dentro de un port pair group, utiliza las tablas de grupo de OVS. Cada tabla dentro de OVS dirige los paquetes hacia diferentes puertos de ingreso dentro de las parejas de puertos en el port pair group, pero la implementación de OVS para realizar esta función, utiliza un hash de la dirección de origen y la MAC para distribuir el tráfico en las tablas de OVS.

Esto quiere decir, que el tráfico desde el mismo origen siempre tendrá el mismo hash, y por lo tanto siempre seguirá la misma ruta dentro de un port pair group. El balanceo de carga en sfc se consigue por lo tanto gracias a OVS y los hash, por lo que difiere del típico balanceo de carga que puede realizar un router convencional, y es más efectivo a más direcciones de origen existan.

3.2.8. Alternativas de implementación y empresas implantando SFC

En la actualidad existen otras alternativas para la implementación de Service Function Chaining, una de las mas conocidas es OpenDayLight [24].

OpenDayLight es un controlador SDN apoyado por varias de las mejores empresas del mundo en tecnologías de red (IBM, Cisco, Microsoft, etc.), este controlador es posible de implementar junto a la infraestructura de OpenStack (figura 3.11[2]), dejando a ODL como proveedor de gestión de red, integrándose así con Neutron, en concreto con ML2 Plugin.

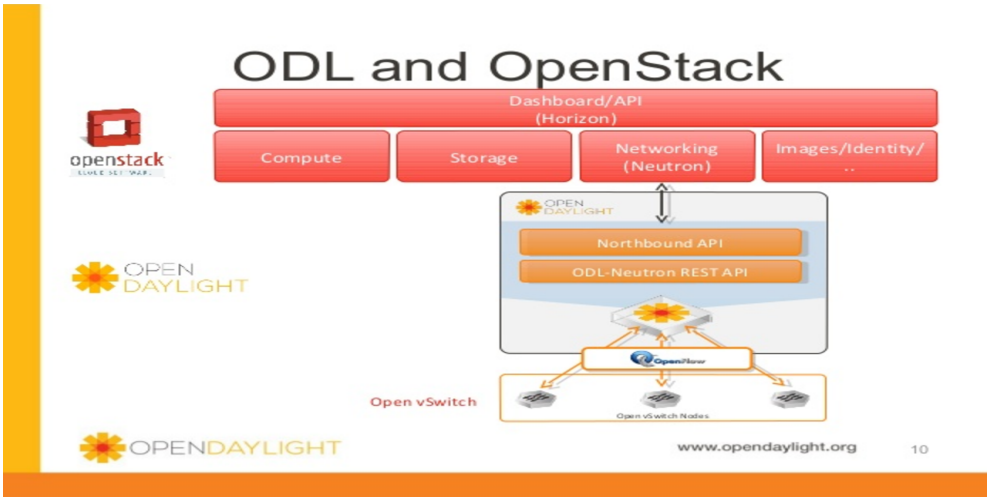


Figura 3.15: OpenDayLight y OpenStack

[2]https://www.slideshare.net/epricec/open-daylight-nluugnovember

OpenDayLight también posee la capacidad de habilitar SFC, por lo tanto se podría ajustar perfectamente a los objetivos de este proyecto, pero requiere de conocimientos técnicos complejos y de un tiempo del cual no se dispone, de ahí la decisión de usar únicamente OpenStack.

En la actualidad existen varias empresas investigando e implantando la tecnología de SFC, entre ellas están:

- ALAXALA
- Hitachi
- Cisco
- NEC
- Alcatel-Lucent

Para trabajos futuros de este proyecto, se propone la integración de ODL como controlador SDN de OpenStack, ya que actualmente, es el proyecto más apoyado en la comunidad SFC, y el que más mejoras está recibiendo, por lo tanto sería altamente recomendable su implantación para líneas de trabajo de futuro. A su vez, para tener un entorno completo, también sería recomendable su implantanción junto con algún tacker que permita la creación de funciones de red con características reales.

Capítulo 4

Diseño de escenarios y evaluación de resultados

En este capítulo se describirán y diseñarán diferentes escenarios de red para la evaluación de Service Function Chaining, seguidamente se verificará la solución y el correcto funcionamiento.

4.1. Escenario 1: Prueba sencilla de SFC

En este escenario se propondrá una solución sencilla, formada por una máquina origen, una máquina destino y una función entre las dos.

El propósito de este escenario, es el de probar la funcionalidad más sencilla de SFC, el encaminamiento de paquetes, para así poder comprobar que todo funciona acorde a la información que contienen las cadenas de servicio y las rutas que deben seguir los paquetes. Este escenario es la base para la creación de escenarios más complejos.

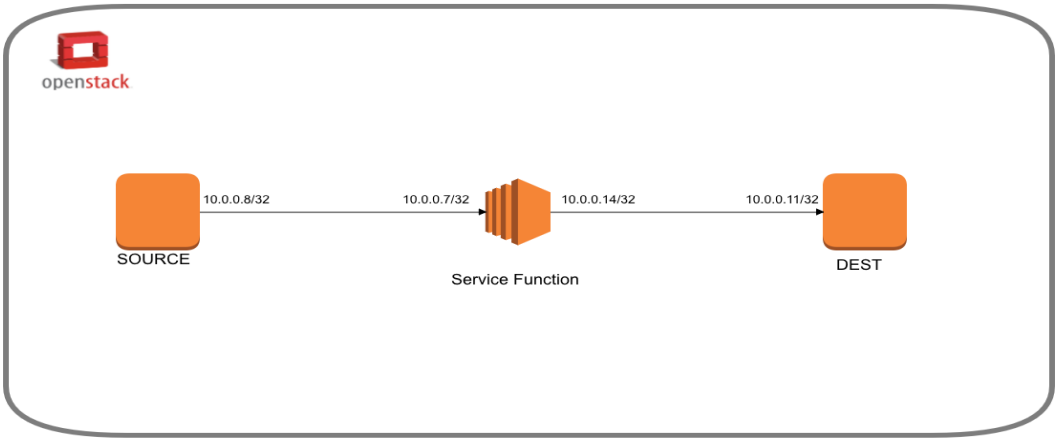


Figura 4.1: Escenario 1

Como se aprecia en la figura 4.1, los paquetes deben salir de la máquina origen, cruzar por la service function y finalmente llegar a la máquina destino.

Para poder desplegar el escenario, primero se han de introducir los siguientes comandos en la consola

- Se utilizan las credenciales del usuario admin en el proyecto demo

```
source openrc admin demo
```

- Se deshabilita el mecanismo anti-spoofing incluido en las últimas versiones de OpenStack, de esta manera se permitirá el forwarding entre puertos de una máquina virtual

```
# Disable port security (This allow spoofing just to make possible
    the IP forwarding)
openstack network set --disable-port-security private
```

- Se crean los puertos para las VMs

```
openstack port create --network private source_port
openstack port create --network private dest_port
openstack port create --network private p1in
openstack port create --network private p1out
```

- Creamos las máquinas virtuales que actuarán como origen, destino y función

```
openstack server create --image cirros-0.3.5-x86_64-disk --flavor
    m1.nano \
    --nic port-id="$(openstack port show -f value -c id source_port)
       " \
     source_vm
openstack server create --image cirros-0.3.5-x86_64-disk --flavor
    m1.nano \
    --nic port-id="$(openstack port show -f value -c id dest_port)"
       \
     dest_vm
openstack server create --image cirros-0.3.5-x86_64-disk --flavor
    m1.nano \
    --nic port-id="$(openstack port show -f value -c id p1in)" \
    --nic port-id="$(openstack port show -f value -c id p1out)" \
     vm1
```

- Se crean las parejas de puertos

```
neutron port-pair-create --ingress=p1in --egress=p1out PP1
```

- Se crean los grupos de pares de puertos

```
neutron port-pair-group-create --port-pair PP1 PG1
```

- Se crea un clasificador para todo el tráfico UDP con origen source_vm y destino dest_vm

```
#Obtain source and destination IP
SOURCE_IP=$(openstack port show source_port -f value -c fixed_ips |
     grep "ip_address='[0-9]*\." | cut -d"'" -f2)
DEST_IP=$(openstack port show dest_port -f value -c fixed_ips |
    grep "ip_address='[0-9]*\." | cut -d"'" -f2)

# UDP flow classifier (UDP traffic)
neutron flow-classifier-create \
    --ethertype IPv4 \
    --source-ip-prefix ${SOURCE_IP}/32 \
    --destination-ip-prefix ${DEST_IP}/32 \
    --protocol udp \
    --logical-source-port source_port \
    FC_udp
```

- Se completa la cadena

```
neutron port-chain-create --port-pair-group PG1 --flow-classifier
    FC_udp PC1
```

- Ahora es necesario configurar la máquina que actúa como función para poder reenviar paquetes de una interfaz a otra, es necesario conectarse bien por ssh o bien por horizon a la consola. En este caso se hará por horizon, en la opción console[1].

Figura 4.2: SSH a la instancia

[1]En este escenario se accederá mediante la consola SSH de horizon para probar su funcionamiento, la manera óptima de hacerlo es a través de IPs flotantes como se verá en los siguientes escenarios

Una vez dentro de la consola, introducimos los siguientes comandos para habilitar el forwarding

```
sudo sh -c 'echo "auto eth1" >> /etc/network/interfaces'
sudo sh -c 'echo "iface eth1 inet dhcp" >> /etc/network/interfaces'
sudo sh -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
sudo /etc/init.d/S40network restart
#Add static routes to reach source through eth0 and destination
    through eth1
sudo ip route add 10.0.0.8 dev eth0
sudo ip route add 10.0.0.11 dev eth1
```

Desde la consola de la instancia de origen, hacemos un traceroute hasta la máquina destino como se ve en la figura 4.3

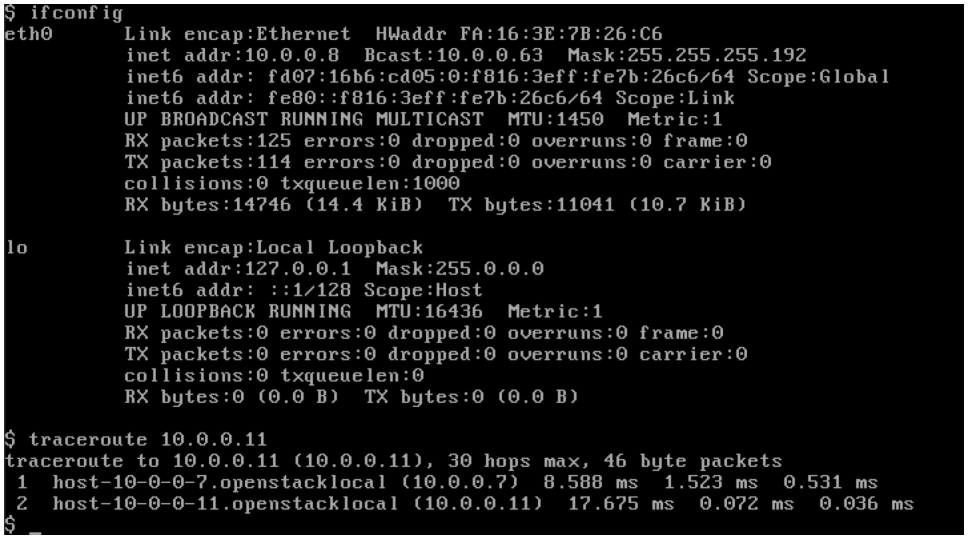


Figura 4.3: Traceroute desde la instancia de origen a la destino

☐	dest_vm	cirros-0.3.5-x86_64-disk	10.0.0.11 fd07:16b6:cd05:0:f816:3eff:fea4:50b7	m1.nano	-	Active
☐	vm1	cirros-0.3.5-x86_64-disk	10.0.0.7 fd07:16b6:cd05:0:f816:3eff:fe91:b0d2 10.0.0.14 fd07:16b6:cd05:0:f816:3eff:fe64:d69a	m1.nano	-	Active
☐	source_vm	cirros-0.3.5-x86_64-disk	10.0.0.8 fd07:16b6:cd05:0:f816:3eff:fe7b:26c6	m1.nano	-	Active

Figura 4.4: Listado de IPs de las máquinas virtuales

Los resultados finales muestran como los paquetes atraviesan la función antes de llegar al destino, y vuelven sin pasar por ésta, ya que no se especificó el parámetro de simetría al crear la cadena. Aunque no se puede apreciar, los paquetes internamente pasan por el SFF antes de llegar a la función y nuevamente vuelven a pasar por el SFF antes de llegar al destino.

En un entorno de red tradicional, donde las máquinas pertenecen todas a la misma subred, los paquetes irían directamente del origen al destino pasando por el router, sin embargo, en este escenario los paquetes pasan por la instancia que forma la cadena (en este caso es solo una). Esto es posible gracias Open vSwitch y al SFF virtualizado, este SFF es el que reenvía los paquetes a las instancias de la cadena gracias a la encapsulación SFC de los paquetes, la cual le permite al SFF decidir a que instancias reenviar.

Se puede concluir que este escenario de red ha funcionado a la perfección, y se aprecia como los paquetes de tráfico UDP (traceroute) siguen el camino indicado. Si se generara otro tipo de tráfico, como puede ser TCP, los paquetes irían directamente del origen al destino.

4.2. Escenario 2: Balanceo de carga en SFC

En este escenario se propondrá una solución para comprobar la funcionalidad de los port pair group con multiples port pairs, el resultado teórico debe dar un balanceo de carga a través de las tablas de grupo de OVS.

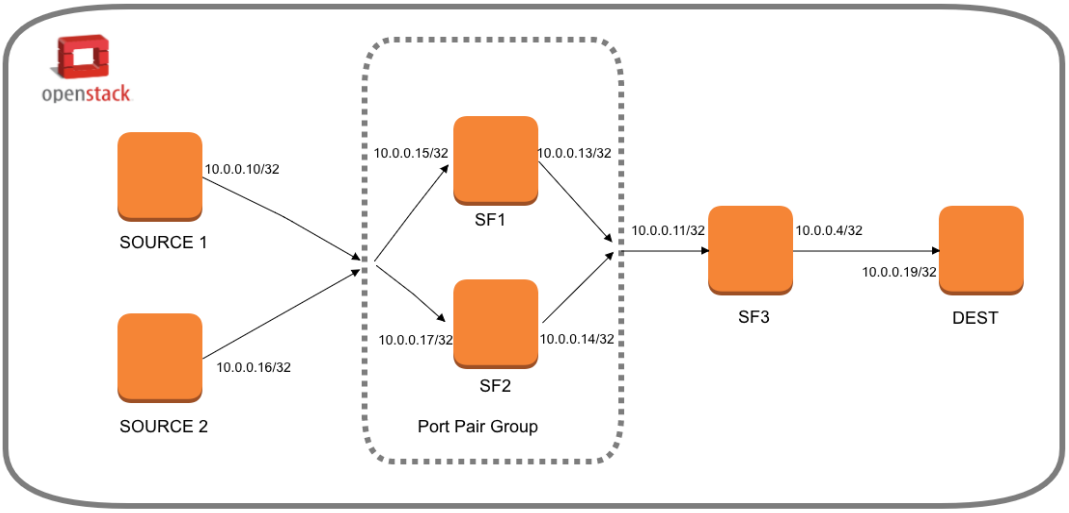


Figura 4.5: Escenario 2

El balanceo de carga es vital en un entorno de producción, donde deben existir varias funciones en paralelo con los mismos propósitos (e.g. firewall), para servir a un gran número de clientes y así no retrasar la entrega de paquetes ni crear ningún tipo de embotellamiento ni congestión.

Para ello se levantarán dos máquinas de origen, tres máquinas que actuarán como funciones, y una máquina destino.

Como se observa en la figura 4.5, los paquetes deben salir desde cualquiera de las dos máquinas de origen, atravesar una de las dos funciones del port pair group, seguir por la función tres y llegar al destino. Para probar este escenario es necesario introducir los siguientes comandos:

- Se utilizan las credenciales del usuario admin en el proyecto demo

```
source openrc admin demo
```

- Se deshabilita el mecanismo anti-spoofing incluido en las últimas versiones de OpenStack, de esta manera se permitirá el forwarding entre puertos de una máquina virtual

```
# Disable port security (This allow spoofing just to make possible
    the ip forwarding)
openstack network set --disable-port-security private
```

- Se añaden las reglas de seguridad para el puerto 80 y 22

```
SECGROUP=$(openstack security group list -f value -c ID --project
    admin 2> /dev/null || echo default)
SECGROUP_RULES=$(openstack security group show "${SECGROUP}" -f
    value -c rules)
if ! echo "${SECGROUP_RULES}" | grep -q icmp
then
    openstack security group rule create --proto icmp "${SECGROUP}"
fi
for port in 22 80
do
    if ! echo "${SECGROUP_RULES}" | grep -q "port_range_max='${port
        }', port_range_min='${port}'"
    then
        openstack security group rule create --proto tcp --dst-port
            ${port} "${SECGROUP}"
    fi
done
```

- Se crean los puertos para las VMs

```
for port in p1in p1out p2in p2out p3in p3out s1 s2 dest_port
do
    openstack port create --network private "${port}"
done
```

- Creamos las máquinas virtuales que actuarán como origen, destino y funciónes

```
openstack server create --image cirros-0.3.5-x86_64-disk --flavor
    m1.nano \
    --nic port-id="$(openstack port show -f value -c id p1in)" \
    --nic port-id="$(openstack port show -f value -c id p1out)" \
     vm1
openstack server create --image cirros-0.3.5-x86_64-disk --flavor
    m1.nano \
    --nic port-id="$(openstack port show -f value -c id p2in)" \
    --nic port-id="$(openstack port show -f value -c id p2out)" \
     vm2
openstack server create --image cirros-0.3.5-x86_64-disk --flavor
    m1.nano \
    --nic port-id="$(openstack port show -f value -c id p3in)" \
    --nic port-id="$(openstack port show -f value -c id p3out)" \
     vm3

openstack server create --image cirros-0.3.5-x86_64-disk --flavor
    m1.nano \
    --nic port-id="$(openstack port show -f value -c id s1)" \
     s1
openstack server create --image cirros-0.3.5-x86_64-disk --flavor
    m1.nano \
    --nic port-id="$(openstack port show -f value -c id s2)" \
     s2
openstack server create --image cirros-0.3.5-x86_64-disk --flavor
    m1.nano \
    --nic port-id="$(openstack port show -f value -c id dest_port)"
        \
     dest_vm
```

- Creamos IPs flotantes para poder acceder a las instancias por SSH

```
SOURCE_FLOATING1=$(openstack floating ip create public -f value -c
    floating_ip_address)
openstack server add floating ip s1 ${SOURCE_FLOATING1}
SOURCE_FLOATING2=$(openstack floating ip create public -f value -c
    floating_ip_address)
openstack server add floating ip s2 ${SOURCE_FLOATING2}
DEST_FLOATING=$(openstack floating ip create public -f value -c
    floating_ip_address)
```

```
openstack server add floating ip dest_vm ${DEST_FLOATING}
for i in 1 2 3; do
    floating_ip=$(openstack floating ip create public -f value -c
        floating_ip_address)
    declare VM${i}_FLOATING=${floating_ip}
    openstack server add floating ip vm${i} ${floating_ip}
done
```

- Se crean las parejas de puertos

```
neutron port-pair-create --ingress=p1in --egress=p1out PPA
neutron port-pair-create --ingress=p2in --egress=p2out PPB
neutron port-pair-create --ingress=p3in --egress=p3out PPC
```

- Se crean los grupos de pares de puertos

```
neutron port-pair-group-create --port-pair PPA --port-pair PPB PGA
neutron port-pair-group-create --port-pair PPC PGB
```

- Se crea un clasificador para todo el tráfico UDP y tráfico HTTP con origen en las máquinas source y destino todas las máquinas en el rango 10.0.0.0/24

```
SOURCE_IP1=$(openstack port show s1 -f value -c fixed_ips | grep "
    ip_address='[0-9]*\." | cut -d"'" -f2)
SOURCE_IP2=$(openstack port show s2 -f value -c fixed_ips | grep "
    ip_address='[0-9]*\." | cut -d"'" -f2)
DEST_IP=$(openstack port show dest_port -f value -c fixed_ips |
    grep "ip_address='[0-9]*\." | cut -d"'" -f2)
neutron flow-classifier-create \
    --ethertype IPv4 \
    --source-ip-prefix 10.0.0.0/24 \
    --destination-ip-prefix 10.0.0.0/24 \
    --protocol tcp \
    --destination-port 80:80 \
    --logical-source-port s1 \
    http1
neutron flow-classifier-create \
    --ethertype IPv4 \
    --source-ip-prefix 10.0.0.0/24 \
    --destination-ip-prefix 10.0.0.0/24 \
    --protocol tcp \
    --destination-port 80:80 \
    --logical-source-port s2 \
    http2
# UDP flow classifier (UDP traffic)
neutron flow-classifier-create \
    --ethertype IPv4 \
```

```
    --source-ip-prefix 10.0.0.0/24 \
    --destination-ip-prefix 10.0.0.0/24 \
    --protocol udp \
    --logical-source-port s1 \
    udp1
neutron flow-classifier-create \
    --ethertype IPv4 \
    --source-ip-prefix 10.0.0.0/24 \
    --destination-ip-prefix 10.0.0.0/24 \
    --protocol udp \
    --logical-source-port s2 \
    udp2
```

- Se completa la cadena

```
neutron port-chain-create --port-pair-group PGA --port-pair-group
    PGB --flow-classifier udp1 --flow-classifier udp2 \
 --flow-classifier http1 --flow-classifier http2 PCA
```

- Ahora es necesario configurar las máquinas que actúan como funciones para poder reenviar paquetes de una interfaz a otra, para ello, se establece una conexión SSH a las instancias a través de las IPs flotantes, para ello se introduce lo siguiente:

```
for i in 1 2 3
do
    ip_name=VM${i}_FLOATING
sshpass -p "cubswin:)" ssh -T cirros@${!ip_name}<<EOF
sudo sh -c 'echo "auto eth1" >> /etc/network/interfaces'
sudo sh -c 'echo "iface eth1 inet dhcp" >> /etc/network/interfaces'
sudo sh -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
sudo /etc/init.d/S40network restart
sudo ip route add ${SOURCE_IP1} dev eth0
sudo ip route add ${SOURCE_IP2} dev eth0
sudo ip route add ${DEST_IP} dev eth1
exit
EOF
done
```

- Se inicia un pequeño servidor que conteste a peticiones HTTP en la máquina destino

```
sshpass -p "cubswin:)" ssh cirros@${DEST_FLOATING} 'while true; do
    echo -e "HTTP/1.0 200 OK\r\n\r\nWelcome to dest-vm" | sudo nc -l
     -p 80 ; done&'
```

Con estos comandos, ya estaría el escenario completamente desplegado para su evaluación.

Una vez creadas las cadenas y el pequeño servidor web, se prueban las rutas de los paquetes (figuras 4.6 y 4.7)

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr FA:16:3E:C7:A7:B5
          inet addr:10.0.0.16  Bcast:10.0.0.63  Mask:255.255.255.192
          inet6 addr: fd67:df97:ddd6:0:f816:3eff:fec7:a7b5/64 Scope:Global
          inet6 addr: fe80::f816:3eff:fec7:a7b5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:205 errors:0 dropped:0 overruns:0 frame:0
          TX packets:147 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:29119 (28.4 KiB)  TX bytes:16982 (16.5 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

$ traceroute 10.0.0.19
traceroute to 10.0.0.19 (10.0.0.19), 30 hops max, 46 byte packets
 1  host-10-0-0-14.openstacklocal (10.0.0.14)  5.418 ms  1.294 ms  0.531 ms
 2  host-10-0-0-4.openstacklocal (10.0.0.4)  2.526 ms  2.682 ms  1.732 ms
 3  host-10-0-0-19.openstacklocal (10.0.0.19)  2.933 ms  0.111 ms  2.757 ms
```

Figura 4.6: Ruta del tráfico UDP desde la fuente 1 al destino

```
$ traceroute 10.0.0.19
traceroute to 10.0.0.19 (10.0.0.19), 30 hops max, 46 byte packets
 1  host-10-0-0-14.openstacklocal (10.0.0.14)  1.300 ms  1.274 ms  1.275 ms
 2  host-10-0-0-4.openstacklocal (10.0.0.4)  2.176 ms  2.338 ms  1.241 ms
 3  host-10-0-0-19.openstacklocal (10.0.0.19)  3.269 ms  2.320 ms  4.164 ms
```

Figura 4.7: Ruta del tráfico UDP desde la fuente 2 al destino

Como se puede observar en este caso, el tráfico UDP desde ambas sources sigue el camino esperado, sin embargo, no existe balanceo de carga en el grupo de pares de puertos, esto probablemente se debe a que las tablas de OVS indican que los paquetes desde ambas IPs y MACs de las fuentes con destino IP DEST, sigan la ruta a través de la SF 10.0.0.14, pero el resultado es totalmente correcto y lo esperado.

Ya que en la cadena se ha especificado que se enrute todo el tráfico con destino 10.0.0.0/24, esto incluye también la IP de la SF3, si se prueba a dirigir tráfico UDP hacia ella (figuras 4.8 y 4.9) se puede observar como en este caso el balanceo de carga si funciona, y los paquetes pasan a través de la IP 10.0.0.8 (SF1) en un caso, y 10.0.0.14 (SF2) en el otro.

```
$ traceroute 10.0.0.4
traceroute to 10.0.0.4 (10.0.0.4), 30 hops max, 46 byte packets
 1  host-10-0-0-14.openstacklocal (10.0.0.14)  1.095 ms  0.699 ms  0.539 ms
 2  host-10-0-0-4.openstacklocal (10.0.0.4)  2.998 ms  1.023 ms  0.915 ms
```

Figura 4.8: Ruta de tráfico UDP desde la fuente 1 a SF3

```
$ traceroute 10.0.0.4
traceroute to 10.0.0.4 (10.0.0.4), 30 hops max, 46 byte packets
 1  host-10-0-0-8.openstacklocal (10.0.0.8)  1.412 ms  0.869 ms  0.597 ms
 2  host-10-0-0-4.openstacklocal (10.0.0.4)  1.290 ms  0.839 ms  0.579 ms
```

Figura 4.9: Ruta de tráfico UDP desde la fuente 2 a SF3

Para probar el camino del tráfico TCP, se procede a hacer un telnet al puerto 80 a la máquina destino (figura 4.10).

```
$ telnet 10.0.0.19 80
HTTP/1.0 200 OK

Welcome to dest-vm
```

Figura 4.10: Telnet desde source2 a dest, tráfico TCP

[2] Una vez hechas todas las pruebas, se puede concluir que el escenario ha sido un éxito y los paquetes han actuado según lo previsto.

El objetivo de este escenario era el de probar los port pair groups y el balanceo de carga, y como se ha podido observar, este balanceo depende de OVS y se han proporcionado ejemplos donde se produce y donde no. Para poder comprobar mejor esta funcionalidad, es recomendable tener un amplio número de máquinas de origen, pero dado que en este proyecto los recursos son limitados, no es posible.

[2]Las imágenes CirrOS no disponen del paquete tcpdump, de ahí que en las capturas no aparezcan los paquetes entrantes y salientes de cada función, aunque el funcionamiento es el correcto, en los próximos ejemplos se dispondrá de una imagen personalizada de CirrOS con el paquete tcpdump, aunque contiene un fallo por el cual no obtiene la dirección del servidor de metadatos y esto hace imposible la conexión por ssh a la ip flotante, únicamente por la consola de horizon.

En un escenario tradicional, no sería posible hacer un balanceo de carga de esta manera, ni el enrutado de paquetes a traves de las funciones, ya que los paquetes van directamente del origen al destino. Para realizar un balanceo de carga, sería necesario el uso de más switches o routers entre las máquinas virtuales, lo que conlleva más gastos y configuraciones técnicas, e incluso, en algún caso extremo, la reestructuración de la red.

4.3. Escenario 3: Distribuidor de red en SFC

En este escenario se propondrá una solución donde una SF actuará como distribuidor de tráfico, y dirigirá los paquetes a una SF u otra en función del tipo de tráfico que pertenezca, hasta alcanzar el destino.

En la figura 4.11 se muestra este escenario, que puede ser perfectamente un ejemplo de producción, donde las tres funciones podrían ser un control parental (tráfico UDP), un firewall (tráfico HTTP) y un sistema IDS (tráfico FTP) en un entorno real. Dado que el alcance este proyecto no cubre la virtualización de las características de funciones de red, se trabajará con máquinas virtuales sin estas funcionalidades.

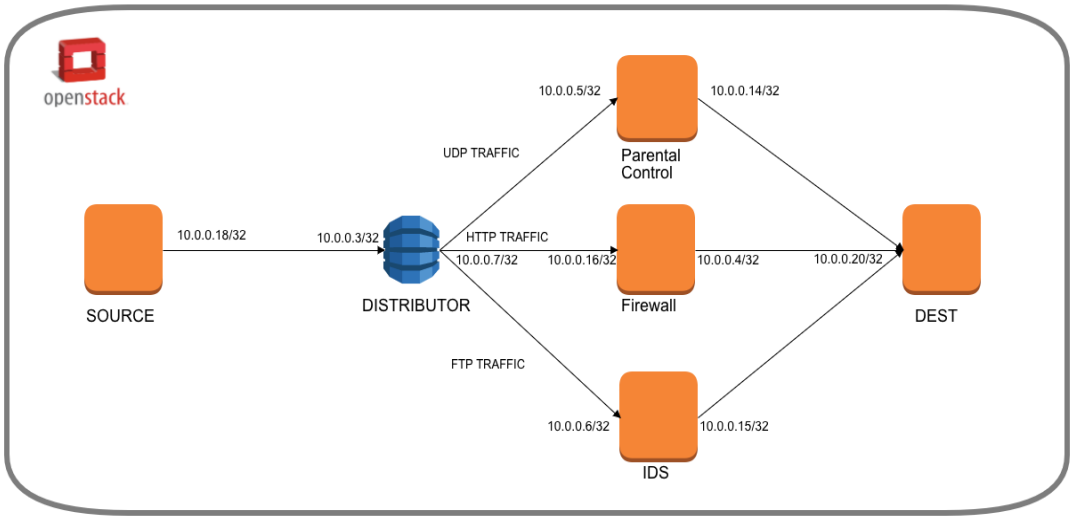


Figura 4.11: Escenario 3

Para desplegar este escenario es necesario introducir los siguientes comandos:

- Se utilizan las credenciales del usuario admin en el proyecto demo

```
source openrc admin demo
```

- Se deshabilita el mecanismo anti-spoofing incluido en las últimas versiones de OpenStack, de esta manera se permitirá el forwarding entre puertos de una máquina virtual

```
# Disable port security (This allow spoofing just to make possible
    the ip forwarding)
openstack network set --disable-port-security private
```

- Se añaden las reglas de seguridad para el puerto 80, 22 y 21

```
SECGROUP=$(openstack security group list -f value -c ID --project
    admin 2> /dev/null || echo default)
SECGROUP_RULES=$(openstack security group show "${SECGROUP}" -f
    value -c rules)
for port in 21 22 80
do
    if ! echo "${SECGROUP_RULES}" | grep -q "port_range_max='${port
        }', port_range_min='${port}'"
    then
        openstack security group rule create --proto tcp --dst-port
            ${port} "${SECGROUP}"
    fi
done
```

- Se crean los puertos para las VMs

```
for port in p1in p1out p2in p2out p3in p3out p4in p4out source_port
     dest_port
do
    openstack port create --network private "${port}"
done
```

- Creamos las máquinas virtuales que actuarán como origen, destino y funciónes

```
openstack server create --image disk-1 --flavor m1.nano \
    --nic port-id="$(openstack port show -f value -c id p1in)" \
    --nic port-id="$(openstack port show -f value -c id p1out)" \
     vm1
openstack server create --image disk-1 --flavor m1.nano \
    --nic port-id="$(openstack port show -f value -c id p2in)" \
    --nic port-id="$(openstack port show -f value -c id p2out)" \
     vm2
openstack server create --image disk-1 --flavor m1.nano \
    --nic port-id="$(openstack port show -f value -c id p3in)" \
    --nic port-id="$(openstack port show -f value -c id p3out)" \
     vm3
 openstack server create --image disk-1 --flavor m1.nano \
```

```
    --nic port-id="$(openstack port show -f value -c id p4in)" \
    --nic port-id="$(openstack port show -f value -c id p4out)" \
     vm4
openstack server create --image cirros-0.3.5-x86_64-disk --flavor
    m1.nano \
    --nic port-id="$(openstack port show -f value -c id source_port)
       " \
     source_vm
openstack server create --image cirros-0.3.5-x86_64-disk --flavor
    m1.nano \
    --nic port-id="$(openstack port show -f value -c id dest_port)"
       \
     dest_vm
```

- Creamos IPs flotantes para poder acceder a las instancias por SSH

```
SOURCE_FLOATING=$(openstack floating ip create public -f value -c
    floating_ip_address)
openstack server add floating ip s1 ${SOURCE_FLOATING1}
DEST_FLOATING=$(openstack floating ip create public -f value -c
    floating_ip_address)
openstack server add floating ip dest_vm ${DEST_FLOATING}
for i in 1 2 3 4; do
    floating_ip=$(openstack floating ip create public -f value -c
       floating_ip_address)
    declare VM${i}_FLOATING=${floating_ip}
    openstack server add floating ip vm${i} ${floating_ip}
done
```

- Se crean las parejas de puertos

```
neutron port-pair-create --ingress=p1in --egress=p1out PPA
neutron port-pair-create --ingress=p2in --egress=p2out PPB
neutron port-pair-create --ingress=p3in --egress=p3out PPC
neutron port-pair-create --ingress=p4in --egress=p4out PPD
```

- Se crean los grupos de pares de puertos

```
neutron port-pair-group-create --port-pair PPA PGA
neutron port-pair-group-create --port-pair PPB PGB
neutron port-pair-group-create --port-pair PPC PGC
neutron port-pair-group-create --port-pair PPD PGD
```

- Se crean tres clasificadores para todo el tráfico UDP, tráfico HTTP y FTP con origen en las máquinas source y destino todas la máquina dest

```
# HTTP Flow classifier (web traffic from source to destination)
```

```
SOURCE_IP=$(openstack port show source_port -f value -c fixed_ips |
     grep "ip_address='[0-9]*\." | cut -d"'" -f2)
DEST_IP=$(openstack port show dest_port -f value -c fixed_ips |
    grep "ip_address='[0-9]*\." | cut -d"'" -f2)
neutron flow-classifier-create \
    --ethertype IPv4 \
    --source-ip-prefix ${SOURCE_IP}/32 \
    --destination-ip-prefix ${DEST_IP}/32 \
    --protocol tcp \
    --destination-port 80:80 \
    --logical-source-port source_port \
    http
# UDP flow classifier (UDP traffic)
neutron flow-classifier-create \
    --ethertype IPv4 \
    --source-ip-prefix ${SOURCE_IP}/32 \
    --destination-ip-prefix ${DEST_IP}/32 \
    --protocol udp \
    --logical-source-port source_port \
    udp
# FTP flow classifier (FTP traffic)
neutron flow-classifier-create \
    --ethertype IPv4 \
    --source-ip-prefix ${SOURCE_IP}/32 \
    --destination-ip-prefix ${DEST_IP}/32 \
    --protocol tcp \
    --destination-port 20:21 \
    --logical-source-port source_port \
    ftp
```

- Se completan las tres cadenas, una para cada tipo de tráfico

```
neutron port-chain-create --port-pair-group PGA --port-pair-group
    PGB \
 --flow-classifier udp PCA
neutron port-chain-create --port-pair-group PGA --port-pair-group
    PGC \
 --flow-classifier http PCB
neutron port-chain-create --port-pair-group PGA --port-pair-group
    PGD \
 --flow-classifier ftp PCC
```

- Ahora es necesario configurar las máquinas que actúan como funciones para poder reenviar paquetes de una interfaz a otra, para ello, se establece una conexión SSH a las instancias a traves de la consola de horizon, para ello se introduce lo siguientes comandos en cada instancia:

```
sudo sh -c 'echo "auto eth1" >> /etc/network/interfaces'
sudo sh -c 'echo "iface eth1 inet dhcp" >> /etc/network/interfaces'
```

```
sudo sh -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
sudo /etc/init.d/S40network restart
sudo ip route add ${SOURCE_IP} dev eth0
sudo ip route add ${DEST_IP} dev eth1
```

Una vez desplegado el escenario, se procede a generar tráfico UDP, HTTP (puerto 80) y FTP (puerto 21) desde la máquina origen.

En la figura 4.13 se aprecian los paquetes con tráfico HTTP mediante una captura de wireshark en el firewall, el resultado es el previsto y los paquetes siguen la ruta que se indicó en la cadena. En la figura 4.14 se muestra una captura wireshark en el sistema IDS, para apreciar como el tráfico FTP sigue la ruta esperada a través del IDS como se indicaba en la cadena. Por último, en la figura 4.12 se muestran un traceroute y una captura de wireshark en el control parental, por donde debe pasar todo el tráfico UDP.

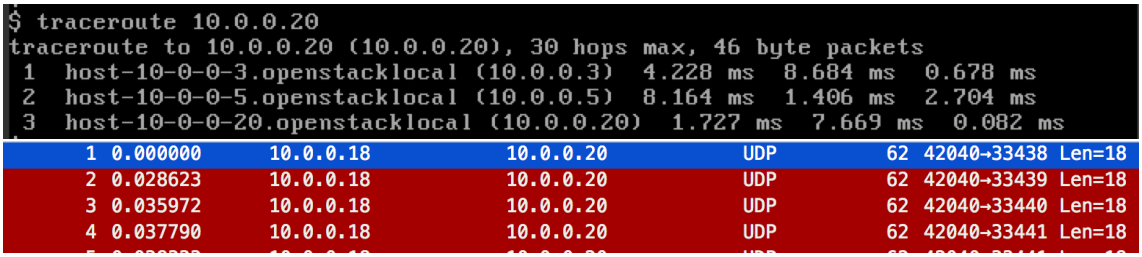


Figura 4.12: Ruta desde source a dest, tráfico UDP y captura wireshark en control parental

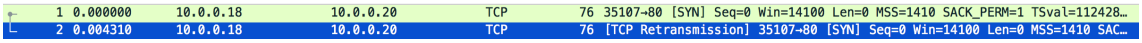


Figura 4.13: Ruta desde source a dest, tráfico HTTP, captura wireshark en firewall

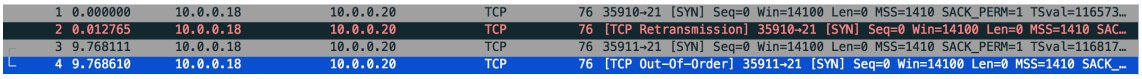


Figura 4.14: Ruta desde source a dest, tráfico FTP, captura wireshark en IDS

Se puede concluir por lo tanto, que este escenario ha sido un éxito, y se cumple el principal objetivo que consistía en probar una máquina que actuará de distribuidor, dirigiendo el tráfico a diferentes funciones según el tipo de tráfico (número de puerto), comprobando así el uso de varias clasificadores y cadenas sobre una misma máquina (distribuidor).

En un entorno tradicional, no sería posible la distribución del flujo de red en función del tipo de tráfico, ya que los routers tradicionales no enrutan tráfico dependiendo del puerto de destino, esto se consigue gracias a las tablas de grupo de open vSwitch, que ofrecen una lectura de paquetes más granular, a su vez, tampoco sería posible la creación de cadenas de servicio, como ya se ha comentado en los dos ejemplos anteriores, por lo cual, se ha demostrado una funcionalidad completamente nueva en las arquitecturas de red que no se podría conseguir sin un entorno virtualizado como el de este proyecto.

Este ejemplo podría ser perfectamente aplicado en un entorno profesional, donde se quisiera separar cada tipo de tráfico y dirigirlo a funciones específicas, en vez de hacerlo pasar por todas a la vez, pudiendo así ahorrar recursos y proporcionar una mayor calidad de servicio.

Capítulo 5

Planificación del trabajo y presupuesto del proyecto

5.1. Planificación del trabajo

5.1.1. Definición de etapas y tareas

La elaboración del proyecto se ha dividido en varias etapas, las cuales se describen a continuación:

- ETAPA 1: Planificación

  Antes de empezar el proyecto, es necesario hacer un estudio de los objetivos, requisitos y las posibles tecnologías que puedan tener cabida en el desarrollo del mismo.

  - Planteamiento: La primera fase del proyecto consiste en plantear una motivación, estudiar los objetivos de este proyecto, su finalidad y la organización de las etapas.
  - Definición de requisitos: Una vez analizados los objetivos principales, se establecen los requisitos técnicos y materiales para la realización de este proyecto.
  - Estudio de plataforma de desarrollo: Se realiza un estudio de que plataforma usar para la consecución del proyecto, las alternativas a ésta, y la decisión basada en términos de presupuesto, requisitos y disponibilidad.
  - Estudio de las tecnologías cloud: Por último, se estudiaron las diferentes soluciones cloud que permiten el uso de SFC, y se toma una decisión en función de la complejidad y el tiempo.

- ETAPA 2: Desarrollo

  2.1 OpenStack

  - Entorno Google Cloud: DevStack corre sobre una máquina virtual en los servidores de Google Cloud, es necesario preparar el entorno, así como configurar los grupos de seguridad para acceder a ella y el servidor VNC.

- Preparar local.conf: Antes de instalar DevStack, es necesario aprender a configurar el local.conf y ver que servicios son necesarios.
- Instalación de DevStack: Una vez preparado el local.conf, se procede a la instalación de DevStack, así como solucionar los constantes problamas que van surgiendo hasta conseguir un entorno estable.
- Uso de OpenStack: Una vez instalado OpenStack, es necesario aprender a usar el mismo, tanto en su versión en Horizon como en la línea de comandos.

2.2 SFC

- Definición de escenarios: Antes de diseñar los escenarios, es necesario analizar los posibles casos, para así poder elegir los que demuestren todas las características de SFC.
- Uso de línea de comandos de SFC: Es necesario aprender a usar los comandos para SFC.
- Despliegue de escenarios: Una vez decididos los escenarios, se procede al despliegue de los mismos en OpenStack.
- Configuración de las instancias: Cuando los escenarios están desplegados y las cadenas creadas, es el momento de solucionar los problemas de routing entre instancias, para ello es necesario aprender a activar el forwarding en linux, así como añadir rutas e interfaces.
- Preparación de scripts: Para automatizar el despliegue, se preparan scripts en lenguaje bash.

- ETAPA 3: Resultado, evaluación y documentación Una vez desplegados todos los escenarios, se procede a su correspondiente evaluación y pruebas de funcionamiento. Seguidamente, una vez todo funcione correctamente, se redacta la memoria.

  - Evaluación de escenarios: Se evalúan los escenarios haciendo diferentes pruebas y se llega a una conclusión de su funcionamiento.
  - Redacción de memoria: Una vez finalizadas las pruebas, se redacta la memoria del proyecto.
  - Preparación de presentación: Preparar una presentación para describir el proyecto

5.1.2. Planificación: Diagrama de Gantt

Una vez definidas las principales tareas, se procede a elaborar un diagrama temporal de Gantt. El periodo se estima teniendo en cuenta el plazo máximo para la entrega del proyecto, el 26 de Septiembre de 2017.

Las jornadas diarias han ido cambiando en función del tiempo disponible del autor, variando entre 4 y 12 horas.

A continuación se muestra el diagrama de Gantt en la tabla 5.1

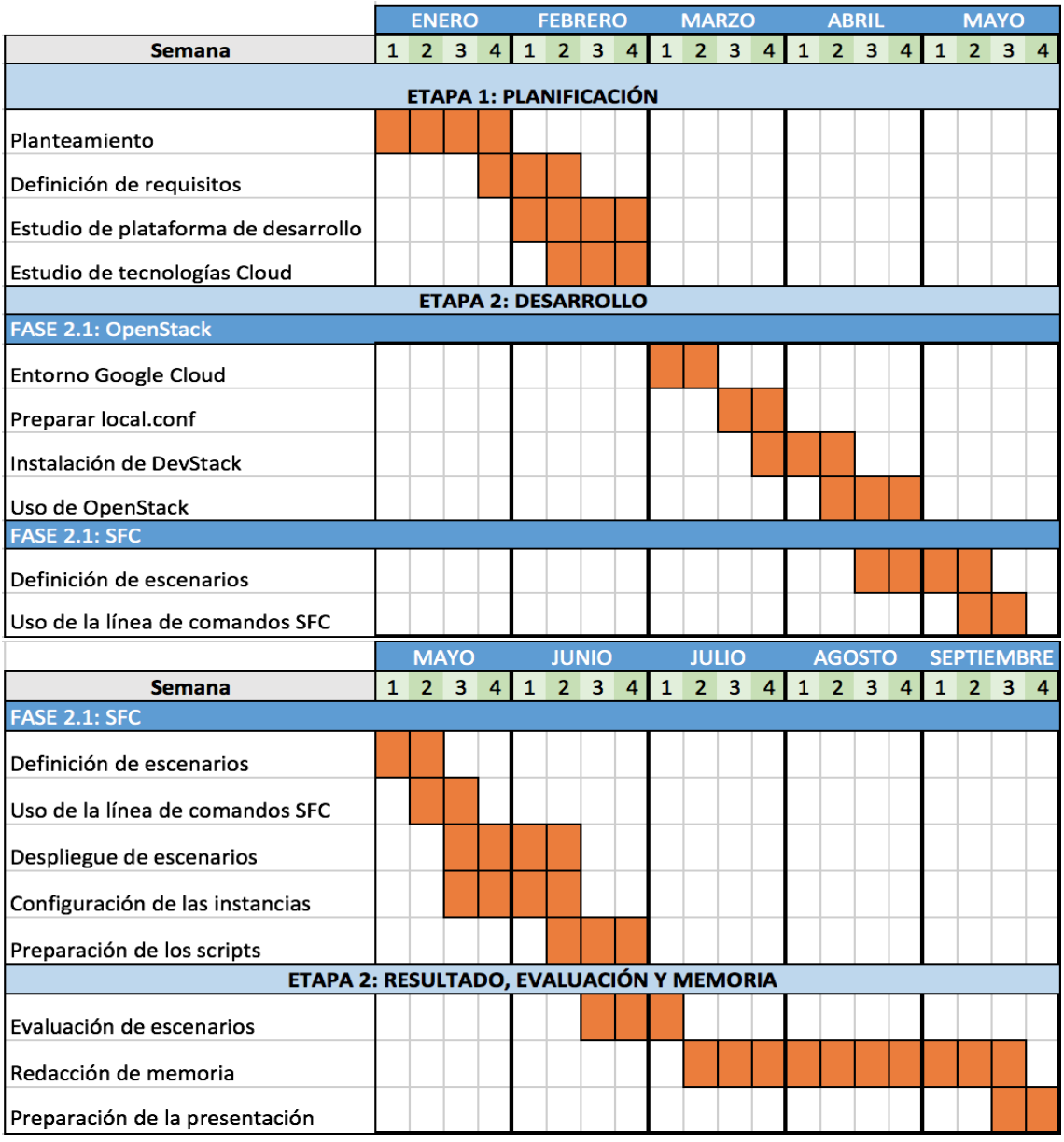


Tabla 5.1: Diagrama de Gantt

5.2. Presupuesto

En esta sección se describen los costes del proyecto, divididos en costes materiales y costes de personal.

5.2.1. Costes materiales

Para los costes materiales se han tenido en cuenta los siguientes:

- Portatil: Macbook Pro

  Es necesario un portátil para poder trabajar en remoto, en este caso se elige uno de la marca Apple por su autonomía, su peso y características.

- Servidor Google Cloud

  La media de horas que se usará el servidor al mes serán unas 100, los costes los podemos comprobar en la figura 3.1

- Material de oficina.

- Alquiler de oficina 20 metros cuadrados en Madrid

  Será necesario un espacio para trabajar en Madrid.

- Gastos de electricidad e Internet

  Hay que considerar también los gastos de electricidad y fibra óptica, ya que se necesita una buena conexión.

5.2.2. Costes de personal

En cuanto a los costes de personal, se tienen en cuenta las horas dedicadas por el autor del proyecto, considerándolo Ingeniero Junior.

Para estimar el salario de un ingeniero Junior, se ha recurrido a ofertas de trabajo en puestos de Ingenieros de sistemas e Ingenieros de tecnologías de virtualización para hacer una estimación.

5.2.3. Costes totales

El presupuesto total del proyecto se muestra en la tabla 5.2:

CONCEPTO	CANTIDAD	COSTE	TIEMPO (meses)	TOTAL €
COSTES MATERIALES				
Portátil	1	1400	9	1400
Google Cloud	100 hr/mes	10,9	9	98,1
Material de Oficina	VARIOS	25	9	25
Alquiler de Oficina	1	300	9	2700
Gastos electricidad	1	25	9	225
Internet fibra óptica	1	50	9	450
				4898,1
COSTES DE PERSONAL				
Ingeniero Junior	1	1750	9	15750
				15750
IVA (21%)				4336,101
COSTES TOTALES				24984,201

Tabla 5.2: Presupuesto

Los costes del proyecto son aproximaciones, teniendo en cuenta que el lugar de trabajo se sitúa en Madrid.

Capítulo 6

Conclusión y trabajos futuros

A continuación se presentan las conclusiones derivadas de la realización de este proyecto de fin de grado. Seguidamente se proponen una serie de mejoras, bien para líneas de proyectos futuros, o bien para personas que quieran continuar la investigación sobre Service Function Chaining.

6.1. Conclusión

En este trabajo de fin de grado se proponen dos objetivos principales, la instalación de un entorno Cloud Open Source llamado OpenStack, y la evaluación práctica de la nueva arquitectura Service Function Chaining.

Gracias al uso de OpenStack, es posible desarrollar una plataforma Cloud completa en cualquier entorno empresarial, pudiendo así prescindir de los servicios que ofrecen otras empresas del sector, o si se plantea de otra manera, es posible la creación de nuevas empresas o líneas de negocio, que ofrezcan su nube pública a otros clientes o servicios públicos.

La solución OpenStack está implantada en la red privada de entornos bancarios como pueden ser Banco Santander, y se encuentra apoyada por empresas como IBM, Intel o Ericsson entre otras, lo que apoya la teoría de que es un proyecto con mucho alcance y futuro. Una de las soluciones interesantes sería la implementación en redes públicas, como pueden ser los hospitales, que necesitan del acceso seguro a todos los documentos e historiales de pacientes a cualquier hora y en cualquier momento, así como la red de policía, o documentos del estado, permitiendo así gestionarse uno mismo todos los aspectos de su plataforma Cloud, así como su seguridad y el hecho de que toda la información está controlada y se sabe su localización exacta.

Por otro lado, el uso de la arquitectura Service Function Chaining, propone una nueva solución en el enrutamiento de paquetes, gracias al uso de las tecnologías SDN y NFV. Gracias a esta solución es posible garantizar una mejor calidad de servicio al cliente final o usuario, ya que se consigue la eficiencia deseada al poder enrutar cada tipo de tráfico a través de la función que se requiera, en vez de hacerlo pasar por todas las funciones, a pesar de que algunas no apliquen ningún tipo de

filtro o análisis necesario al mismo.

El beneficiado en el uso de estas dos soluciones, es tanto el proveedor como el usuario final, el primero gracias al uso eficiente de sus recursos, el menor coste de actualización y despliegue de funciones, y el ahorro de recursos económicos. A su vez, el usuario se ve beneficiado en una mayor calidad de servicio ofrecida por parte del proveedor a la hora de recibir contenido.

La dificultad técnica de este proyecto se puede considerar elevada, en relación al nivel de experiencia que se exige al autor, un ingeniero Junior. Son necesarios conocimientos en diferentes áreas, como pueden ser el uso de los numerosos comandos de Ubuntu, el conocimiento de las redes de paquetes y su funcionamiento interno, la teoría de nuevas tecnologías como Open vSwitch o OpenFlow y conocimientos para el desarrollo de scripts en bash.

Finalmente, a pesar de la complejidad de algunas tareas, los objetivos del proyecto han sido completados. Además el trabajo realizado durante estos meses le ha servido al autor para fortalecer y aprender nuevas habilidades técnicas. Se puede concluir que la valoración final es muy positiva.

6.2. Mejoras y desarrollos futuros

Para concluir con la documentación de este proyecto se proponen varias mejoras para líneas de proyectos futuros o mejoras.

- **Mejoras en el balanceo de carga.**

  Dado que el balanceo de carga se hace a través de Open vSwitch, se podría buscar la manera de hacer el balanceo a través de otro mecanismo, o no usar hash.

- **Desarrollo de funciones reales.**

  Las funciones vistas en este proyecto no tienen las funcionalidades de los dispositivos reales, es por ello que sería buena idea virtualizar las funciones reales en las instancias.

- **Uso de cloud-init.**

  Mediante cloud-init es posible iniciar las instancias con ciertas características ya instaladas.

- **Acceso a las instancias desde la internet pública.**

  Para acercarse más a un entorno real, sería conveniento poder acceder a la cadena de funciones desde fuera del entorno OpenStack.

- **Integración de Opendaylight.**

  Opendaylight es el controlador SDN que más está siendo apoyado en la comunidad SFC, por lo tanto sería apropiada su integración con OpenStack.

Capítulo 7

Anexos

En este capítulo se propondrán varios anexos para completar el trabajo de fin de grado.

7.1. Anexo 1: Instalación de Google Cloud

En esta sección se explicarán los pasos para levantar una instancia en Google Cloud y acceder a ella a través de un servidor VNC [28]

Creando un nuevo proyecto

Visitar la consola de desarrolladores, logearse, crear un proyecto mediante el botón Create Project. Navegar en el panel de la izquierda *Compute - Compute Engine - VM instances*.

Creando una nueva instancia

Clicar en el botón Create Instance para acceder al formulario de creación. Elegir un nombre, el tipo de distribución y la zona que más cerca se encuentre para disminuir la latencia 3.1

Instalando un servidor VNC

Primero se introduce el siguiente comando

```
1 sudo apt-get install gnome-core
```

Seguidamente se instala un servidor VNC

```
1 sudo apt-get install vnc4server
```

Ahora es necesario editar el siguiente archivo

```
1 nano .vnc/xstartup
```

Este archivo debe tener esta forma

```
#!/bin/sh

# Uncomment the following two lines for normal desktop:
unset SESSION_MANAGER
# exec /etc/X11/xinit/xinitrc

#[ -x /etc/vnc/xstartup ] && exec /etc/vnc/xstartup
#[ -r $HOME/.Xresources ] && xrdb $HOME/.Xresources
#xsetroot -solid grey
#vncconfig -iconic &
#x-terminal-emulator -geometry 80x24+10+10 -ls -title "$VNCDESKTOP
    Desktop" &
#x-window-manager &

metacity &
gnome-settings-daemon &
gnome-panel &
```

Instalando un cliente VNC

Es necesario descargar cualquier cliente VNC, en este caso se optó por VNC Viewer.

Abrir el Firewall

El primer paso es añadir el tag a la instancia, para ello es necesario ir al boton add tags en la descripción (figura 7.1).

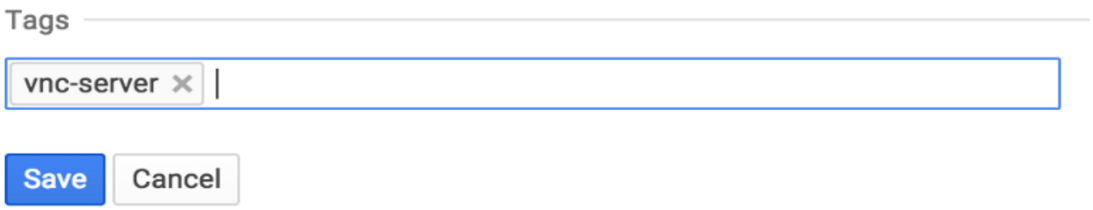


Figura 7.1: Añadir tag a la instancia

Navegar hasta la configuración de la red por defecto *Compute - Compute Engine - Network* y clicar en default. Ahora hay que añadir una nueva regla al firewall (figura 7.2).

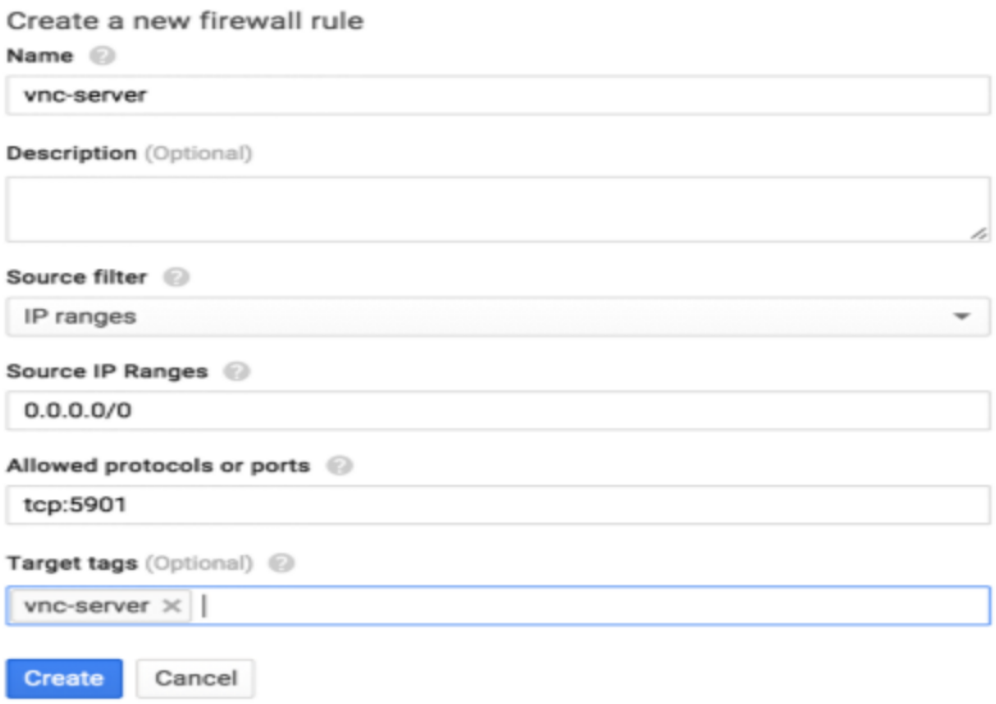


Figura 7.2: Añadir regla al firewall de Google Cloud

Conectando al servidor VNC

Arrancar el servidor VNC

```
1 vncserver
```

Ahora simplemente hay que colocar la IP pública de la instancia y conectarse al puerto 5901 mediante VNC Viewer (figura 7.3).

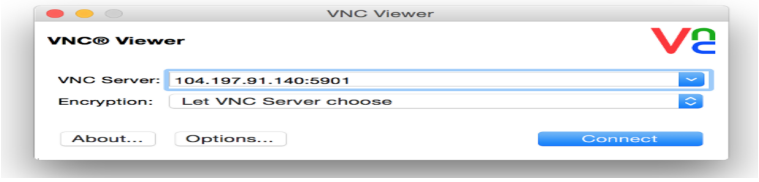


Figura 7.3: Conectar mediante VNC Viewer

7.2. Anexo 2: Scripts de despliegue de escenarios

Los scripts creados para la automatización de los despliegues de escenarios se pueden conseguir del repositorio GitHub del autor

```
git clone https://github.com/javibr/Openstack-scripts
```

Es necesario hacer unstack cada vez que se quiera desmontar un escenario y montar uno nuevo.

Los scripts existentes son los siguientes:

- **sfc_scenario.sh** representa el primer escenario
- **scenario2.sh** representa el segundo escenario
- **scenario3.sh** representa el tercer escenario
- **options.sh** contiene las opciones de credenciales, SSH keys y grupos de seguridad
- **route.sh** provee acceso a las instancias

Bibliografía

[1] Comunidad OpenStack, *Arquitectura de OpenStack.*

https://docs.openstack.org/newton/install-guide-rdo/common/get-started-conceptual-architecture.html

[2] Comunidad OpenStack, *Web oficial de OpenStack*

https://www.opestack.org

[3] NIST, *The NIST Definition of Cloud Computing*

http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

[4] SearchTelecom , *Ahorro de costes SDN*

http://searchtelecom.techtarget.com/photostory/2240177303/SDN-survey-offers-snapshot-of-provider-plans-and-vendor-issues/5/SDN-architecture-expected-to-bring-simplicity-cost-savings

[5] Qi Zhang · Lu Cheng · Raouf Boutaba , *Cloud computing: state-of-the-art and research challenges, p.9*

http://ai2-s2-pdfs.s3.amazonaws.com/6109/3ca3afcc72f04bebbbad7b9fd98d09438122.pdf

[6] Comunidad OpenStack , *Horizon: The OpenStack Dashboard Project*

https://docs.openstack.org/horizon/latest/

[7] Comunidad OpenStack , *OpenStack Compute (nova)*

https://docs.openstack.org/nova/latest/

[8] Comunidad OpenStack , *Keystone, the OpenStack Identity Service*

https://docs.openstack.org/keystone/latest/

[9] Comunidad OpenStack , *Welcome to Neutron's documentation!*

https://docs.openstack.org/neutron/latest/

[10] Comunidad OpenStack , *Welcome to Swift's documentation!*

https://docs.openstack.org/swift/latest/

[11] Comunidad OpenStack , *Cinder, the OpenStack Block Storage Service*

https://docs.openstack.org/cinder/latest/

[12] Comunidad OpenStack , *Welcome to Glance's documentation!*

https://docs.openstack.org/glance/latest/

[13] Comunidad OpenStack , *Welcome to the Heat documentation!*

https://docs.openstack.org/heat/latest/

[14] Comunidad OpenStack , *Welcome to Ceilometer's documentation!*

https://docs.openstack.org/ceilometer/latest/

[15] sdxcentral , *Understanding the SDN Architecture*

https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/

[16] Cisco , *Openflow section*

https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html

[17] sdxcentral , *An Overview of NFV Elements*

https://www.sdxcentral.com/nfv/definitions/nfv-elements-overview/

[18] Internet Engineering Task Force (IETF), J. Halpern, Ed. Ericsson C. Pignataro, Ed. Cisco , *Service Function Chaining (SFC) Architecture*

http://www.rfc-editor.org/rfc/rfc7665.txt

[19] sdxcentral , *What is Network Service Chaining? Definition*

https://www.sdxcentral.com/sdn/network-virtualization/definitions/what-is-network-service-chaining/

[20] Comunidad OpenStack , *Service function chaining*

https://docs.openstack.org/newton/networking-guide/config-sfc.html

[21] Boletín Oficial del Estado , *Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal.*

https://www.boe.es/buscar/doc.php?id=BOE-A-1999-23750

[22] Ronald van der Pol , *D1.2 OpenFlow*

https://www.surf.nl/binaries/content/assets/surf/en/knowledgebase/2012/RoN-2011-D1.2.pdf

[23] Comunidad OpenStack , *Ask Openstack*

https://ask.openstack.org/en/questions/

[24] Comunidad OpenStack , *What is an OpenDaylight Controller? AKA: OpenDaylight Platform*

https://www.sdxcentral.com/sdn/definitions/sdn-controllers/opendaylight-controller/

[25] Comunidad OpenStack , *Command List*

https://docs.openstack.org/python-openstackclient/latest/cli/command-list.html

[26] Comunidad OpenStack , *local.conf*

https://docs.openstack.org/devstack/latest/configuration.html

[27] Comunidad OpenStack , *DevStack*

https://docs.openstack.org/devstack/latest/

[28] Aditya Choudhary , *Graphical user interface (GUI) for Google Compute Engine instance*

https://medium.com/google-cloud/graphical-user-interface-gui-for-google-compute-engine-instance-78fccda09e5c

[29] Louis Fourie (lfourie) , *SFC port pair group loadbalancer not working Edit*

https://bugs.launchpad.net/networking-sfc/+bug/1604427

[30] Fujitsu Laboratories Ltd. , *Using Open vSwitch for service function chain (SFC) and SFC proxy to realize NFV*

http://openvswitch.org/support/ovscon2015/17/1310-nakagawa.pdf